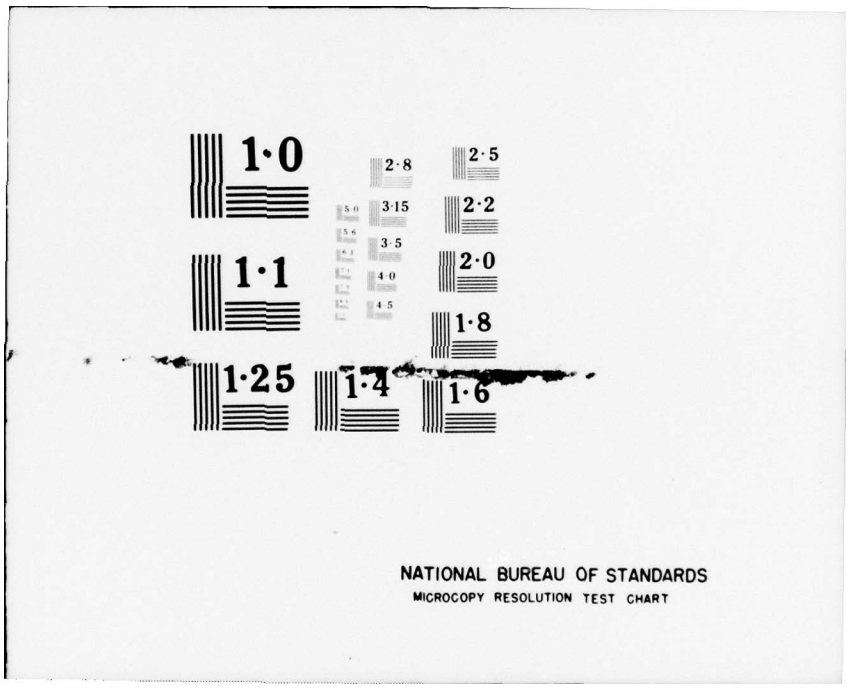
AD-A040 828 GAERTNER (W W) RESEARCH INC STAMFORD CONN F/G 5/1
IMPACT OF STRUCTURED PROGRAMMING STANDARDS ON SMALL GOVERNMENT --ETC(U)
MAY 77 W W GAERTNER F30602-76-C-0390
UNCLASSIFIED RADC-TR-77-162-VOL-2 NL

1 OF 3
ADA
040828


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

AD A040828

VOL A040771

12 NW

RADC-TR-77-162, Volume II (of two)
Final Technical Report
May 1977

IMPACT OF STRUCTURED PROGRAMMING STANDARDS ON SMALL GOVERNMENT CONTRACTORS

W. W. Gaertner Research Inc

DDC
RECEIVED
JUN 21 1977
C

Approved for public release; distribution unlimited.

AD No.
DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report contains a large percentage of machine-produced copy which is not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: Michael Landes

MICHAEL LANDES
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-77-162, Volume II (of two)	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): IMPACT OF STRUCTURED PROGRAMMING STANDARDS ON SMALL GOVERNMENT CONTRACTORS. Volume II.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Aug 76 — Feb 77.
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Dr. Wolfgang W. Gaertner	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0390
9. PERFORMING ORGANIZATION NAME AND ADDRESS: W. W. Gaertner Research Inc, 205 Saddle Hill Road, Stamford CT 06903	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55811411
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIM), Griffiss AFB NY 13441	12. REPORT DATE: May 1977
	13. NUMBER OF PAGES: 234
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DDC RELEASED JUN 21 1977 C

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Same

18. SUPPLEMENTARY NOTES
RADC Project Engineer:
Michael Landes (ISIM)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Structured Programming, Small Business, Small Contractor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A study of the impact of the structured programming standards on small government contractors. How they can reasonably meet such requirements within the normal environment of the small contractor.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

388544

II.1 General Comments on Volume II

Volume II contains the detailed comments prepared during a line-by-line review of the entire Structured Programming Series. Analyses, clarifications and recommendations are provided wherever the original text is not self-explanatory. In addition, examples for the implementation of the various requirements in the Structured Programming Series are presented in all cases where the effort required was minimal. The purpose in providing these examples was to demonstrate that many aspects of the proposed SP standards can be readily implemented by the Small Contractor. In most cases, three different versions of the same example are given, one implemented on a typical in-house minicomputer system, another on a typical Time-Sharing System used by the Small Contractor, and a third one using a large computer at a typical Government Installation.

The examples show that while there are variations in the ease with which the various requirements can be met on different computer systems, there do not appear to exist any substantial difficulties in conforming to the proposed Structured Programming Standards via any of the computing media typically used by the Small Contractor.

II.2 Completeness of Coverage

Every line of all 15 volumes of the Structured Programming Series has been reviewed, and notations are provided even on paragraphs which are entirely "clear".

II.3 Detailed Comments on the Impact of the Guidelines and Standards of the Structured Programming Series on Smaller Government Contractors

II.3.1 Structure of Chapter II.3

The structure of Chapter II.3 follows the Statement of Work of the Contract such that the comments on a specific subject area are presented in a single chapter, even if it is mentioned in several different volumes and chapters of the Structured Programming Series. The reference to the proper location within the Structured Programming Series is provided by a notation preceding a given comment of recommendation, as follows:

> SPS V-3.1.1, e.g., contains the comments and recommendations on the material presented in Chapter 3.1.1 of Volume V of the Structured Programming Series.

II.3.2 Programming Support Library (PSL)

The subject of the Programming Support Library is addressed in Volumes I, Programming Language Standards; IV, Data Structuring Study; V, Programming Support Library, Functional Requirements; VI, Programming Support Library, Program Specifications; X, Chief Programmer Team Operations Description; XII, Training Materials; XIII, Final Report; XIV, Software Tool Impact; and XV, Validation and Verification Study of the Structured Programming Series. This Chapter II.3.2 provides detailed comments, examples and recommendations concerning any chapter within the Structured Programming Series where the subject of the Programming Support Library is mentioned.

SPS I-2.3 Programming Support Library

a) This is a brief review of the PSL which is expanded upon in Volumes V and VI.

SPS IV-4.3.2 Data Structure Usage Accounting

Clear

This subsection is duplicated in the Programming Support Library description.

SPS V Programming Support Library (PSL) Functional Requirements

SPS V-1.0 Introduction

Clear

SPS V-2.0 Programming Support Libraries

SPS V-2.1 Definition

Clear

SPS V-2.2 Purpose

Clear

SPS V-2.3 General Description

Clear

Batch operations are not used frequently by the small contractor. They seem more applicable to the batch program development procedures of a large company

which, according to Volume IX of the SPS may have an average turnaround time of 4 hours per program change! In fact, undue emphasis has been placed on the Batch Environment at least from the standpoint of the Small Contractor, since this approach to program development is both exceedingly inefficient and generally not available to the Small Contractor who will usually use either on-line time-share services or on-line in-house minicomputers. However, if the Small Contractor is required to use the Batch Environment it should not create a problem, since the customer would then presumably make his own computer (with full PSL facilities) available to the Small Contractor.

Otherwise, however, the Small Contractor will work mostly in the Terminal Environment on a time-share system or on an in-house minicomputer.

SPS V-3.0 Detailed Functional Requirements

SPS V-3.1 Source Data Maintenance

SPS V-3.1.1.1 Basic Requirements

a) Data File Storage

Clear

The small contractor is, of course, aware of the advantages of storing data on line. Whether in-house, on a time sharing system, or accessing a Government or other customer computer the key items such as program source code, program object modules, program load modules, job control data and test data are stored on random access (core, disk) devices. The other statements, textual data and other data are retained on mountable volumes (tapes). The main purpose of the separation is to decrease storage costs by having the less often altered information on lower cost media.

SPSV-3.1.1.b) Data Access

The above mentioned storage techniques provide the "timely" access to the data required as a basic PSL condition.

SV-3.1.1.c) Library backup capability

In house backup of PSL files is readily accomplished by the small contractor. The minimum storage medium available is paper tape. However, in-house systems usually have one or more of the following capabilities available for storage: Cassette tape, floppy disk, magnetic tape, disk pack, or punched cards. As can be seen the off-line storage media cover the full spectrum available in the industry. The time sharing environment offers the same broad range of storage devices to the user. And Government installations can also offer many of the above mentioned media. The ease with which the files on disk can be backed up on magnetic tape can be seen in Figure II.3.2-1. It is a terminal session on an in-house computer that has selectively transferred disk files to tape. Time shared facilities offer the same ease of media interaction, as shown in Figure II.3.2-2. The contractor's experiences with Government computer systems has shown that they sometimes are not as sophisticated. The capability to transfer files from disk to tape may be more cumbersome, but it is available.

```
.R PIP
*MT0:*.*=*.FIG/X
*MT0:/L
18-NOV-76
B55T01.FIG      1  18-NOV-76
B55T02.FIG      1  18-NOV-76
B55T03.FIG      1  18-NOV-76
3 FILES, 3 BLOCKS
*MT0:B55T4T.FOR=B55T4T.FOR
*MT0:B55CS2.*=B55CS2.*/X/M:1
*MT0:B55GEN.*=B55GEN.*/X/M:1
*MT0:*.*=*.SAM/X/M:2000
*MT0:/L
18-NOV-76
B55T01.FIG      1  18-NOV-76
B55T02.FIG      1  18-NOV-76
B55T03.FIG      1  18-NOV-76
B55T4T.FOR      1  18-NOV-76
B55CS2.SAV     12  18-NOV-76
B55GEN.SAV     15  18-NOV-76
B55GEN.FOR      1  18-NOV-76
B55EDS.SAM      3  18-NOV-76
B55SED.SAM      3  18-NOV-76
B55CDE.SAM      3  18-NOV-76
BXTMP .SAM      1  18-NOV-76
B55SS7.SAM      3  18-NOV-76
B55SS8.SAM      2  18-NOV-76
B55SS9.SAM      1  18-NOV-76
B55CD1.SAM      2  18-NOV-76
B55CD2.SAM      3  18-NOV-76
B55CD3.SAM      3  18-NOV-76
B55CD5.SAM      1  18-NOV-76
B55CD6.SAM      2  18-NOV-76
B55CD7.SAM      3  18-NOV-76
B55CD8.SAM      2  18-NOV-76
B55CD9.SAM      1  18-NOV-76
B55CDA.SAM      2  18-NOV-76
23 FILES, 67 BLOCKS
```

Figure II.3.2-1. Backing up disk files on magnetic tape using a typical in-house minicomputer.

```
14.01.32 >MOUNT TAPE X2234 AS 282 RINGIN

14.02.01 >DEV 282 ATTACHED
>T REW

14.07.22 >TAPE SKIP 16

14.08.39 >TAPE SCAN
SWITCHES FORTRAN
SWM37A   DATA
SWM88    DATA
SWM3720  DATA
SWM3738  DATA
SWM372   DATA
SWM3713  DATA
*EOF

14.08.44 >TAPE DUMP * DATA
DUMPING...
SWN37B   DATA
SWN88B   DATA
SWN88C   DATA
SWN3738B DATA
SWN374B  DATA
SWN374C  DATA

14.09.05 >TAPE WTM

14.09.16 >TAPE DUMP SWNVERB FORTRAN

14.10.02 >TAPE WTM

14.10.13 >DET 282
```

Figure II.3.2-2. Backing up disk files on magnetic tape using a typical commercial time sharing computer.

Recovery of files stored offline is as easy as backing up the files both in-house (Figure II.3.2-3) and on a time sharing system (Figure II.3.2-4). Government computer systems generally offer the same facilities.

BEST AVAILABLE COPY

```
.R PIP
*MT0:/L
18-NOV-76
B55T01.FIG      1  18-NOV-76
B55T02.FIG      1  18-NOV-76
B55T03.FIG      1  18-NOV-76
B55T4T.FOR      1  18-NOV-76
B55CS2.SAV     12  18-NOV-76
B55GEN.SAV     15  18-NOV-76
B55GEN.FOR      1  18-NOV-76
B55EDS.SAM      3  18-NOV-76
B55SED.SAM      3  18-NOV-76
B55CDE.SAM      3  18-NOV-76
BXTMP .SAM      1  18-NOV-76
B55SS7.SAM      3  18-NOV-76
B55SS8.SAM      2  18-NOV-76
B55SS9.SAM      1  18-NOV-76
B55CD1.SAM      2  18-NOV-76
B55CD2.SAM      3  18-NOV-76
B55CD3.SAM      3  18-NOV-76
B55CD5.SAM      1  18-NOV-76
B55CD6.SAM      2  18-NOV-76
B55CD7.SAM      3  18-NOV-76
B55CD8.SAM      2  18-NOV-76
B55CD9.SAM      1  18-NOV-76
B55CDA.SAM      2  18-NOV-76
23 FILES, 67 BLOCKS
**.*=MT0:*.FIG/X
*B55GNB.FOR=MT0:B55GEN.FOR
*B55GNB.*/L
18-NOV-76
B55GNB.FOR      1  18-NOV-76
 582 FREE BLOCKS
*
```

Figure II.3.2-3. Recovery of files from magnetic tape on a typical in-house minicomputer.

```
14.14.23 >MOUNT TAPE X2234 AS 282

14.15.03 >DEV 282 ATTACHED
>TAPE REWIND

14.20.14 >TAPE SKIP 17

14.21.39 >TAPE LOAD SWN374C DATA

14.22.39 >LISTF SWN374C DATA
SWN374C  DATA       P     53

14.22.52 >TAPE SKIP

14.23.07 >TAPE LOAD * *
LOADING...
SWNVERB  FORTRAN
*EOF

14.23.49 >LISTF SWNVERB FORTRAN
SWNVERB  FORTRAN    P     235

14.24.20 >DET 282
```

Figure II.3.2-4 Recovery of files from magnetic tape on a typical commercial time sharing computer.

SPS V-3.1.1.d) Data Maintenance

A basic PSL requirement for source data maintenance, can be accomplished via the system editor and/or utility routines usually available to a time sharing user and on an in-house or Government computer.

SPS V-3.1.1.d.1 Addition of Units of Data to a File

Adding one or more units of data to a file is a basic function of any editor. The procedure is to select the location where the data is to be added. This may be any position desired in the file. Then the editor command to input the data is invoked and the data typed. Of course,

the appropriate delimiter characters must contain the data. They may vary from editor to editor but generally any character may be used that does not occur in the input string. Figure II.3.2-5 shows how a unit of data is inserted into a file using the in-house computer of the contractor.

```
.R E
*EBSAMPLE.DAT$$
*R$$
*3A$V$$
   2.221  37.637   5.838  96.291  25.208
*I  11.111$V$$
  11.111   2.221  37.637   5.838  96.291  25.208
*G5.$=C4.444   5.$V$$
  11.111   2.221  37.637   4.444   5.838  96.291  25.208
*B/L$$
   0.082   0.412   1.730   6.674  24.472
  86.765   0.341  21.157  23.878  52.854
  11.111   2.221  37.637   4.444   5.838  96.291  25.208
  84.626  80.885  23.679  14.110  71.548
```

Figure II.3.2-5. Adding one or more units of data to a file via a typical in-house editor.

First, the editor program is invoked with the system command R E. Next the file is identified and a prefix edit command is used to identify the operation desired. In this case an EB, meaning edit back, is employed. This command automatically creates a copy of the existing file on disk, and assigns the output to the file name issued in the command. The dollar signs are command deliminators which are typed by the user as "Alt Mode" key on a terminal. The next line R$$ invokes the editor to read the file into its buffer. There are 25 basic edit commands available. The appropriate ones are now selected to insert the new data. The editor being described employs a common pointer technique. The pointer is advanced 3 lines by the instruction 3A$V$$ which also requests that the line be displayed with the V or verify command. A unit of data is now inserted at the

beginning of the line as the next line entered by the user. To show the versatility of the editor a second unit of data is inserted in the middle of the same line. Any number of insertions may be made during a single edit session or any of the other editor commands may be invoked to alter the file as desired.

The edit capability of the time sharing system accessed by this contractor is more versatile than the in-house editor. It has 37 distinct commands:

AGAIN
BACKSPACE
BLANK
BOTTOM
BRIEF
CHANGE
CSS
DELETE
DOWN/NEXT
ERASE
FILE
FIND
GET
GOTO
HEX
INPUT
INSERT
LINENO
LOCATE
X, Y, XX, and YY Macros
OVERLAY
PRINT
PUT/PUTD
QUESTION MARK (?)
QUIT
REPLACE
SAVE
SBRIEF
SERIAL
SETBREAK
SETCHAR
TABDEF
TABSET
TOP
UP
VERIFY
ZONE

Figure II.3.2-6 contains an edit session to add one or more units of data to a file. One command, EDIT SAMPLE DATA, is issued to invoke the editor, read the file into the edit buffer and assign the output file. The output file is automatically assigned the same name as the input file but the editor allows reassignment of the output file if desired. The same operation as the previous example displayed is accomplished with this demonstration. The commands are slightly different. First, the pointer is advanced to the third line via the NEXT 3 command. The abbreviated form of the command is employed, N 3. The CHANGE command will insert a unit of data into the beginning of the line. The slashes (/) are delimiters for the string to be altered. Actually, no string is shown between the first two delimiters, so the editor places the data at the beginning of the line. The CHANGE command automatically allows verification of the change by immediately printing the new line. Next, a unit of data is inserted into the middle of the line. Exit from the editor is via the FILE command which saves the new file on a permanent storage disk.

```
08.34.14 >EDIT SAMPLE DATA
EDIT:
>N 3
   2.221  37.637   5.838  96.291  25.208
>C //  11.200/
  11.200   2.221  37.637   5.838  96.291  25.208
>C /5./7.777   5./
 11.200   2.221  37.637   7.777   5.838  96.291  25.208
>TOP 99
   0.082   0.412   1.730   6.674  24.472
  86.765   0.341  21.157  23.878  52.854
 11.200   2.221  37.637   7.777   5.838  96.291  25.208
 84.626  80.885  23.679  14.110  71.548
>FILE

08.36.42 >
```

Figure II.3.2-6. Adding one or more units of data to a file via a typical commercial time sharing editor.

The Government computer system most recently accessed by the contractor was a CDC 6600 system that supported the CDC time sharing system called INTERCOMM. It will be considered as a typical Government installation. The editor has only 12 commands:

BYE, END
FORMAT
CREATE
LINE = TEXT
EDIT
SAVE
LIST
ADD
DELETE
RESEQ
TEXT REPLACEMENT
RUN

To add one or more units of data to a file the following commands are necessary (Figure II.3.2-7 contains the actual session). First, file identification must be established between the permanent storage disk and the user. This is accomplished with the ATTACH command. The system would assign a default local file name. However, in this example the local file name of SMP was assigned. Next, the editor is invoked to run with the command EDITOR to the system. Edit mode is identified by the double period, indicating the editor is ready to accept any valid command. The initial instruction to the editor is to read the file into its buffer and sequence it. This editor employs a line sequence number technique to reference the file. The default sequence numbers are 100 for line 1, 110 for line 2, 120 for line 3 and so forth. To add one or more units of data to the file as in the above examples the editor is told to reference the third line with the command L, 120. This lists the line with sequence number 120. Next we insert a unit of data at the beginning of the line. Care must be taken when doing this: One option is to retype the entire line which has the obvious drawback of possibly introducing errors. The second option is to issue the text replacement command. The text replacement command requires two data strings to operate. The first string must be a valid set of characters or a single character. The

```
COMMAND- ATTACH,SMP,SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- EDITOR

..E,SMP,S
..L,120

120=    1.221   37.637    5.838   96.291   25.208

../ 1./=/11.300    1./
  1 CHANGES

..L,120
120=   11.300    1.221   37.637    5.838   96.291   25.208

../5.8/=/3.333   5.8/
  1 CHANGES

..L,120
120=   11.300    1.221   37.637    3.333    5.838   96.291   25.208

..L,A

100=    0.082    0.412    1.730    6.674   24.472
110=   86.765    0.341   21.157   23.878   52.854
120=   11.300    1.221   37.637    3.333    5.838   96.291   25.208
130=   84.626   80.885   23.679   14.110   71.548

..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLEDATA,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEDATA
 CT CY= 002     0000128 WORDS.:
COMMAND-
```

Figure II.3.2-7. Adding one or more units of data to a file using a typical Government computer.

previous example allowed a null first string, this editor does not. The second string may be the replacement string or null, essentially causing an information delete. Another peculiarity of the instruction is that it will search the entire line for string 1 and replace it with string 2 unless an option column delimiter extension is included. It is more common to find this method as a global option in other editors. The unique string "blank, one, period" is selected for the first parameter and the replacement string 11.300 1. is entered as the second parameter. The two fields are delimited by slashes (/) and separated by an equal sign. To verify the change, a List Line 120 is issued. An optional verify could have been appended to the text replacement command but this would have required the user to respond either Y or N indicating yes or no to invoke the change. Insertion of a unit of data into the middle of the line is performed similarly to the above data placement using the text replacement command. The final form of the new line is then printed by issuing the List command. Exit from the editor is preceded with a Save of the new file into a local mode. A new local file reference name must be employed, in this case x. Since significant storage reduction may be obtained by storing the file without the sequence numbers, the N option is appended to remove the sequence numbers. Actual exit from the editor is accomplished by typing the command BYE. The SAVE instruction merely moved a copy of the file from core to a temporary storage device. If the programmer were to logout from the system at this time all of the changes made to the file would be lost. It is necessary to make a permanent copy of the file with yet another command. The CATALOG command is used to make the file permanent. The parameters to the instruction are x, the local file reference, SAMPLEDATA, the permanent file name and ID=GAERTNER, an identifier the user associates with the file that can be used for all related files in his system. There are additional options available but they are not pertinent at this time.

A brief introduction to the computing capabilities of the small contractor has been given in the preceding three examples. The reader will find that the commercial time sharing system editor has anticipated many of the actions the user is required to explicitly perform in either commercially available mini or large scale computing systems. The in-house minicomputer offers more capability and ease of application than the large computer system. The following examples will further confirm the above statements.

SPS V-3.1.1.d.2 Delete One or More Units of Data

The in-house minicomputer system available to the small contractor offers the capability to delete one or more units of data from a file as shown in Figure II.3.2-8. The editor is invoked and the file is accessed in the same manner as previously described. Note that this example shows how commands can be combined onto one line issuing single ALT modes between commands and a double ALT mode to terminate the command string. Again the third line is accessed. The two data items added to the file will be deleted with the GET command. The first unit is set equal to a null field. Also, the second unit of data is removed in a similar manner. Exiting from the editor is identical to the preceding example for the in-house computer.

```
.R E
*EBSAMPLE.DAT$R$3A$V$$
  11.111    2.221   37.637    4.444    5.838   96.291   25.208
*G  11.111$=C$V$$
    2.221   37.637    4.444    5.838   96.291   25.208
*G4.444     $=C$V$$
    2.221   37.637    5.838   96.291   25.208
*B/L$$
    0.082    0.412    1.730    6.674   24.472
   86.765    0.341   21.157   23.878   52.854
    2.221   37.637    5.838   96.291   25.208
   84.626   80.885   23.679   14.110   71.548
*EX$$
```

Figure II.3.2-8. Deleting one or more units of data from a file using a typical in-house minicomputer.

The commercial time sharing vendor's editing system may also readily delete one or more units of data. Figure II.3.2-9 contains the terminal session that deletes the data. Again the editor is invoked and the pointer advanced to the third line. The units of data entered previously are now deleted with two CHANGE commands. Exit from the editor is accomplished with the FILE instruction.

```
10.02.13 >EDIT SAMPLE DATA
EDIT:
>N 3
  11.200   2.221  37.637   7.777   5.838  96.291  25.208
>C / 11.200//
   2.221  37.637   7.777   5.838  96.291  25.208
>C /  7.777//
   2.221  37.637   5.838  96.291  25.208
>T#P 9
   0.082   0.412   1.730   6.674  24.472
  86.765   0.341  21.157  23.678  52.854
   2.221  37.637   5.838  96.291  25.208
 84.626  80.885  23.679  14.110  71.548
>FILE

10.04.56 >
```

Figure II.3.2-9. Deleting one or more units of data from a file using a typical commercial time sharing system available to a small contractor.

The editor of a Government installation computer is also able to delete one or more units of data from a file. Employing the same rather tedious access method the file is edited and the pointer is advanced to the third line. The text replacement command is issued for both units of data to delete their occurrence in the file. Then the file is retained on a permanent device as seen in Figure II.3.2-10.

```
COMMAND- ATTACH,SMP,SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO.= 002
COMMAND- EDITOR

..E,SMP,S
..L,120

120=  11.300    1.221   37.637    3.333    5.838   96.291   25.208

../11.300  /=//
  1 CHANGES

..L,120
120=    1.221   37.637    3.333    5.838   96.291   25.208

../3.333   /=//
  1 CHANGES

..L,120
120=    1.221   37.637    5.838   96.291   25.208

..L,A

100=    0.052    0.412    1.730    6.674   24.472
110=   86.765    0.341   21.157   83.878   52.854
120=    1.221   37.637    5.838   96.291   25.208
130=   84.626   80.885   23.679   14.110   71.548

..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLEDATA,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEDATA
 CT CY= 003     0000128 WORDS.:
COMMAND-
```

Figure II.3.2-10. Deleting one or more units of data from a file using a typical Government installation computer.

SPS V-3.1.1.d.3 Replacing One or More Units of Data

It should be obvious to the reader that the third basic requirement of data maintenance in the programming support library, namely replacing one or more units of data with new units of data can be accomplished in an almost identical manner to the second requirement. The only difference is that instead of a null replacement string, the string is filled with actual data. Figure II.3.2-11 shows how replacement would be accomplished on the in-house computer. Next, Figure II.3.2-12 contains the instructions to accomplish the task on a commercial time sharing system. Finally, Figure II.3.2-13 shows how replacing of data is accomplished on a typical Government computer.

```
.R E
*EB$AMPLE.DAT$R$3A$V$$
   2.221   37.637    5.838   96.291   25.208
*G37.637$=C33.345$V$$
   2.221   33.345    5.838   96.291   25.208
*G96.2$=C22.9$V$$
   2.221   33.345    5.838   22.991   25.208
*B/L$$

   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   33.345    5.838   22.991   25.208
  84.626   80.885   23.679   14.110   71.548
*EX$$

.
```

Figure II.3.2-11. Replacing one or more units of data with new units of data using a typical in-house minicomputer.

```
10:46.23 >EDIT SAMPLE DATA
EDIT:
>N 3
    2.221   37.637    5.838   96.291   25.208
>C /37.637/22.234/
   2.221   22.234    5.838   96.291   25.208
>C /25.20/55.59/
   2.221   22.234    5.838   96.291   55.598
>TOP 9
    0.082    0.412    1.730    6.674   24.472
 86.765    0.341   21.157   23.878   52.854
   2.221   22.234    5.838   96.291   55.598
 84.626   80.885   23.679   14.110   71.548
>FILE

10.47.38 >
```

Figure II.3.2-12. Replacing one or more units of data with new units of data using a commercial time sharing system editor.

```
COMMAND- ATTACH,SMP,SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO.= 003
COMMAND- EDITOR

..E,SMP,S
..L,120

120=    1.221   37.637    5.838   96.291   25.208

../1.22/=/3.99/
  1 CHANGES

..L,120
120=    3.991   37.637    5.838   96.291   25.208

../.291/=/.629/
  1 CHANGES

..L,120
120=    3.991   37.637    5.838   96.629   25.208

..L,A

100=    0.082    0.412    1.730    6.674   24.472
110=   86.765    0.341   21.157   23.878   52.854
120=    3.991   37.637    5.838   96.629   25.208
130=   84.626   80.885   23.679   14.110   71.548

..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLEDATA,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEDATA
 CT CY= 004    0000128 WORDS.:
COMMAND-
```

Figure II.3.2-13. Replacing one or more units of data with new units of data using a typical Government computer.

SPS V-3.1.1.d.4 Add, Delete and/or Replace One or More Lines of Source Code

The fourth requirement of data maintenance in the PSL deals with changing source code. The requirements are to add, delete and/or replace one or more lines of source data. It has just been shown how the three example editors available to the small contractor can perform the three operations within a line of data. The capability to operate on an entire line is also available. Figure II.3.2-14 illustrates how the in-house editor would add one or more lines of data. The example inserts one line of source before line 3 and two lines of source after line 3. The example shows a portion of FORTRAN code. The process is described as follows: Initialization of the editor is done with the R E command to the operating system. Next, the file is accessed, read, and advanced to the third line which is displayed. The insert instruction then places a line of code in front of the third line and the command -L is issued to list the line. Now the following line is referenced and two lines of source are placed in front of it. Finally, the editor pointer is moved to the beginning of the file with a B and the /L instruction lists the entire file.

```
.R E
 *EBSAMPLE.FOR$R$3A$V$$
        BESTFT=(A+A2)/2.
*I      SAVSUM=A+A2
$-L$$
        SAVSUM=A+A2
*A$I    HAVG=SAVSUM +TRANS +REGIN
        TGRET=HAVG /PI
$B/L$$
        A=C(I)
        A2 =C(I+1)
        SAVSUM=A+A2
        BESTFT=(A+A2)/2.
        HAVG=SAVSUM +TRANS +REGIN
        TGRET=HAVG /PI
        GAUSIN=FGAUS( BESTFT )
*EX$$

.
```

Figure II.3.2-14. Adding one or more lines of source code to a file using a typical in-house editor.

The typical time sharing system editor accomplishes the addition of one or more lines of source code via similar, though not identical, instructions. The same insertions as above are made in Figure II.3.2-15. One line is added before the third line and two lines are added after it.

```
11.40.07 >EDIT SAMPLE FORTRAN
EDIT:
>N 3
         BESTFT=(A+A2)/2.
>I       SAVSUM=A+A2
>UP
         SAVSUM=A+A2
>N 2
         GAUSIN=FGAUS( BESTFT )
>I
INPUT:
         HAVG=SAVSUM +TRANS +REGIN
         TGRET=HAVG /PI

EXIT:
>TOP 9
         A=C(I)
         A2 =C(I+1)
        SAVSUM=A+A2
         BESTFT=(A+A2)/2.
         HAVG=SAVSUM +TRANS +REGIN
         TGRET=HAVG /PI
         GAUSIN=FGAUS( BESTFT )
>FILE

11.43.12 >
```

Figure II.3.2-15. Adding one or more lines of source code to a file using a typical commercial time sharing editor.

The Government computer installation example is shown in Figure II.3.2-16. After file accesses and editor initialization, the lines may be added. Since the programmer knows that the third line has sequence number 120 and the second line is sequence number 110, selection of an integer number between 110 and 120 will place a new line of source between the two existing lines. 115 is selected and the new line is added. To insert lines after the original third line, 130 is selected as a starting point. The ADD instruction is issued with the first line number to add, 121, and the second parameter is the line number increment value. To stop the addition process an equal sign (=) is entered.

```
COMMAND- ATTACH,F,SAMPLEFORT,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- EDITOR
..E,F,S

..115=   SAVSUM=A+A2
A,121,1

121=              HAVG=SAVSUM +TRANS +REGIN
122=              TGRET=HAVG /PI
123=

..L,A

100=              A=C(I)
110=              A2 =C(I+1)
115=              SAVSUM=A+A2
120=              BESTFT=(A+A2)/2.
121=              HAVG=SAVSUM +TRANS +REGIN
122=              TGRET=HAVG /PI
130=              GAUSIN=FGAUS( BESTFT )

..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLEFORT,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEFORT
 CT CY= 002     0000128 WORDS.:
COMMAND-
```

Figure II.3.2-16. Adding one or more lines of source code to an existing file using a Government-installation computer.

The file is then listed and a cleanup is performed.

In the previous two editors the only limitation to the number of lines that may be added to a file at one time is the amount of memory that is available to the user. This is not a true limitation though, because both editors can handle files of unlimited size by using special options. It should be pointed out that these options only need to be invoked when the files are excessively large. The in-house computer at this contractor, e.g., is able to handle source files in excess of 500 lines. The time sharing system must resort to special options when the file exceeds one million characters. The Government computer's editor, employing a line sequence pointer system, must invoke special options when more than nine lines are added in one place.

SPS V-3.1.1.d.4.b Deleting Source Lines

Deleting lines of data is also within the capability of an editor. The in-house computer has the KILL instruction (K) to delete source lines as shown in Figure II.3.2-17. The three lines entered in the add example will be deleted. The first is removed by first referencing the line and then issuing the K command. The pointer is then moved to the next line to be removed and an integer 2 is prefixed to the K indicating that two lines should be deleted.

```
.R E
*EBSAMPLE.FOR$R$3A$V$$
        SAVSUM=A+A2
*K$A$V$$
        HAVG=SAVSUM +TRANS +REGIN
*2K$V$$
        GAUSIN=FGAUS( BESTFT )
*B/L$$
        A=C(I)
        A2 =C(I+1)
        BESTFT=(A+A2)/2.
        GAUSIN=FGAUS( BESTFT )
*EX$$
```

Figure II.3.2-17. Deleting one or more lines of source data using a typical in-house minicomputer.

The time sharing system editor happens to have Delete as the instruction mnemonic to remove lines of source code. Figure II.3.2-18 contains an example to show how lines are deleted from a source file. The abbreviation of the delete command is employed, namely DE.

```
11.49.07 >EDIT SAMPLE FORTRAN
EDIT:
>D 3
        SAVSUM=A+A2
>DE
>N
        HAVG=SAVSUM +TRANS +REGIN
>DE 2
>P
        GAUSIN=FGAUS( BESTFT )
>T#P 9
        A=C(I)
        A2 =C(I+1)
       BESTFT=(A+A2)/2.
       GAUSIN=FGAUS( BESTFT )
>FILE

11.51.02 >
```

Figure II.3.2-18. Deleting one or more lines of data from a source file via a commercial time sharing system's editor.

It should be pointed out in the above figure that the abbreviation D means down where the top of the file is assumed to be the first line and the bottom of the file is the last line. The D 3 instruction advances the pointer to the third line of the file.

The Government computer's editor also has Delete as the instruction mnemonic to remove lines or a line from a file. Figure II.3.2-19 displays the series of instructions to remove the lines

of code put in by the previous add example. Since the file was retained without the sequence numbers, each of the lines added will now have new numbers. The reader will recall that the line numbers 115, 121 and 122 were added. When the file is edited this time, their numbers will be 120, 140 and 150 respectively, which is verified at the beginning of the edit session by listing the file. The abbreviation of the Delete command in this editor is D. There is no Down command in this editor. Two forms of the Delete instruction are available. The first form, applicable to a single line, has only one parameter on the delete. The second form gives the command a range of line sequence numbers to delete. It is used to remove the pair of lines that were added.

```
COMMAND- ATTACH,F,SAMPLEFORT,ID=GAERTNER

 PF CYCLE NO.= 002
COMMAND- EDITOR
..E,F,S

..L,A

100=              A=C(I)
110=              A2 =C(I+1)
120=              SAVSUM=A+A2
130=              BESTFT=(A+A2)/2.
140=              HAVG=SAVSUM +TRANS +REGIN
150=              TGRET=HAVG /PI
160=              GAUSIN=FGAUS( BESTFT )

..D,120

..D,140,150

..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLEFORT,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEFORT
 CT CY= 003    0000128 WORDS.:
COMMAND-
```

Figure II.3.2-19. Deleting one or more lines of source code using a typical Government computer.

SPS V-3.1.1.d.4.c Replacing One or More Lines of Source

Replacement of one or more lines of data may also be accomplished with the three representative editors. An analysis of the Replace function shows that it is a simple combination of the Delete and Add. An example of how this is performed on a small contractor's in-house computer is contained in Figure II.3.2-20. The example shows line one being replaced by a new line and lines two and three are replaced by four lines. The X instruction causes the data being entered to replace the information contained in the file from the present position of the pointer to the next end of line. One line may be replaced with any number of lines. To replace multiple consecutive lines in a file, they must first be removed and then the new lines inserted. The example shows the basic operation of replacing one line and two lines.

```
.R E
*EB$AMPLE.DAT$R$B$/L$$
        A=C(I)
        A2 =C(I+1)
        BESTFT=(A+A2)/2.
        GAUSIN=FGAUS( BESTFT )
*X      BASEA=CDEC +BSUMA
$V$$
        A2 =C(I+1)
*2K$I   BASEAA =( WSTRL + ESTRL + YQAST )/ PI
        BESTFT =BESTFT + BASEA +2.*BASEAA
$V$$
        GAUSIN=FGAUS( BESTFT )
*B/L$$
        BASEA=CDEC +BSUMA
        BASEAA =( WSTRL + ESTRL + YQAST )/ PI
        BESTFT =BESTFT +BASEA +2.*BASEAA
        GAUSIN=FGAUS( BESTFT )
*EX$$

.
```

Figure II.3.2-20. Replacing one or more lines of source code via a typical small contractor's in-house computer.

The commercial time sharing editor has REPLACE as the mnemonic to perform the specified function. It is identical to the minicomputer editor Replace operation in that it replaces only one line, but any number of lines may replace the single line. To replace multiple lines that are consecutive they must first be deleted and the new information entered. Figure II.3.2-21 is a demonstration of how replacement is performed.

```
13.03.34 >EDIT SAMPLE FORTRAN
EDIT:
>P 9
         A=C(I)
        A2 =C(I+1)
        BESTFT=(A+A2)/2.
         GAUSIN=FGAUS( BESTFT )
>T#P
         A=C(I)
>R       BASEC=CCSS + CODVAL
*P
         A2 =C(I+1)
>DE 2
>I
INPUT:
        BASEAA=( WSTRL + ESTRL +YQAST )/ PI
         BESTFT=BESTFT +BASEC +2.*BASEAA

EDIT:
>T#P 9
         BASEC=CCSS +CODVAL
        BASEAA=( WSTRL +ESTRL +YQAST )/ PI
         BESTFT=BESTFT +BASEC +2.*BASEAA
         GAUSIN=FGAUS( BESTFT )
>FILE

13.05.57 >
```

Figure II.3.2-21. Replacing one or more lines of source code using a commercial time sharing editor.

The Government computer editor does not have a named Replace command. However, the function can be implemented. To replace one line of code the line sequence number is typed, followed by an equal sign and then the replacement line. Replacing more than one line of code is accomplished by deleting the existing code and then adding the new code. Another method would be to employ the ADD command with its optional line sequence increment parameter set appropriately and to include the overwrite option. This second method is not recommended because it is possible to overwrite code that was not intended to be replaced and it is also possible to remove fewer lines of code than desired. The method of deleting and adding is more direct and less prone to error. Figure II.3.2-22 is presented to illustrate how code is replaced on a Government computer.

```
COMMAND- ATTACH,F,SAMPLEFORT,ID=GAERTNER

 PF CYCLE NO.= 003
COMMAND- EDITOR
..E,F,S

..L,A

100=              A=C(I)
110=              A2 =C(I+1)
120=              BESTFT=(A+A2)/2.
130=              GAUSIN=FGAUS( BESTFT )

..100=            BASEC=CCDC +CODVAL

..D,110,120

..A,110,10

110=              BASEAA=(WSTRLY +ESTRLY +YQAST) / PI

120=              BSETFT=BESTFT +BASEC +2.*BASEAA
ADD WONT REPLACE OR BYPASS LINES

..L,A

100=              BASEC=CCDC +CODVAL
110=              BASEAA=(WSTRLY +ESTRLY +YQAST) / PI
120=              BESTFT=BESTFT +BASEC +2.*BASEAA
130=              GAUSIN=FGAUS( BESTFT )

..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLEFORT,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEFORT
 CT CY= 004    0000128 WORDS.:
COMMAND-
```

Figure II.3.2-22. Replacing one or more lines of source code using a typical Government computer.

SPS V-3.1.1.d.5 Organize Data for Easy Reference

SPS V-3.1.1.d.5.a Automatic Generating of Line Sequence Numbers for Data Added to the Library.

Automatic line sequence number generation is not available from the minicomputer editor of this small contractor. Two methods are discussed to implement sequencing of a file. The first method would be to modify the system editor. This would make serialization automatic. The disadvantage of such an approach is that the editor would then be unique to the installation. The second alternative would be to write a simple program to serialize files. The files to be serialized would then be processed by the program as required. While this approach is not automatic, it will provide files with sequence numbers. A program to accomplish the above could be developed and provided to the small contractor by the Government as an element of a library to support structured programming.

The typical commercial time sharing editor has automatic generation of line sequence numbers. There is also an editor command to specify what the starting sequence number should be and the increment value. An additional option is to turn off the file serialization which is beneficial for program data files that might contain data in columns 73-80. Colums 73 through 75 hold a three character identification and columns 76 through 80 contain a five digit number. Figure II.3.2-23 displays a printout of a data file that includes the line sequence numbers while Figure II.3.2-24 shows how the line sequence numbers are appended to a source code listing.

As previously mentioned, the Government computer installation editor requires the line sequence numbers to operate on a file. The preceding examples show this sequence number at the beginning of the lines printed by the editor. When the file is stored the field is moved to suffix the lines. The user may optionally store the file with or without the line sequence numbers. Retaining the file with the sequence numbers adds considerably to the amount of storage required for the file, especially for FORTRAN source files which normally utilize only a small portion of the line for actual instructions. The intervening blanks between the instruction and the sequence numbers are not compressed which accounts for most of the additional storage.

SPS V-3.1.1.d.5.b Regeneration of Line Sequence Numbers

The in-house minicomputer does not have regeneration capabilities for line sequence numbers. However this should be a capability of the program library to be supplied by the Government.

Each time a file is edited on the commercial time sharing system, the file is automatically resequenced. Figure II.3.2-25 shows how the file shown in Figure II.3.2-23 changes when new data is added. The sequence numbers are automatically adjusted to accommodate the new lines.

BEST AVAILABLE COPY

```
SAMPLE   DATA      P      ID=GAERTNER          12/14/76    09.51.17          PAGE 1
         VP/CSS    ---          NATIONAL CSS, INC. (STAMFORD DATA CENTER)

 67.772    6.114  26.738    5.401  91.762    1.962  85.912  97.815  13.682BS100010
  1.754   87.389   8.546   64.769  11.706   87.312  18.518  25.301  85.146BS100020
 83.169   32.702  47.698   91.808  21.662    3.697  27.227  30.085  35.467BS100030
 42.035   33.011  19.752   21.411  50.698   11.491  12.662  72.554  21.364BS100040
 75.197   58.908  76.672   29.861  89.117   65.951  93.652  68.350  67.228BS100050
 58.224   24.289  51.714   91.687  84.696   82.991  35.682  67.175  81.911BS100060
 86.863   84.156  22.901   79.996  73.872   23.268  74.758  39.132  61.973BS100070
 19.649   60.140  83.998   62.734  20.419   57.910  63.687  60.930  92.400BS100080
  0.030    4.580  73.210   98.039  29.341   93.703  98.147  45.554  90.005BS100090
 30.040   70.194  50.807   73.098  81.321   30.045  48.379  19.866  83.783BS100100
 23.903   89.376  21.128   22.384  44.152   63.452  83.346  29.009  23.938BS100110
 82.547   79.842  36.122   98.158  63.849   99.674  23.402  43.347  49.467BS100120
  6.674   94.841   8.984    0.331  21.133   23.814  52.689   1.809  36.648BS100130
  3.613   91.942  18.534   94.627  40.956   84.094  35.964  58.939  29.958BS100140
 49.296   26.151  13.240   44.081  45.327   75.229  43.433  83.538  10.328BS100150
 10.126   67.803  15.679   83.850  61.991   17.293  45.839  19.398   3.838BS100160
 48.450   56.156   0.986   99.416  91.517   49.860  75.505   4.288  46.187BS100170
 38.527   15.480  46.138   37.508   9.804   21.254  39.289  44.450  13.102BS100180
 78.566   53.445  13.627    0.760  81.914   84.650  70.670  62.173  37.003BS100190
 62.466   41.764  88.394   54.486  31.368   97.835   4.702  47.696  43.860BS100200
```

Figure II.3.2-23. Sample data file showing line sequence numbers produced on a typical commercial time sharing computer.

```
SAMPLE   FORTRAN  F      ID=GAERTNER          11/24/76    14.13.18          PAGE 1
         VP/CSS   ---          NATIONAL CSS, INC. (STAMFORD DATA CENTER)


      SUBROUTINE MISPTS                                                  SAM00010
C                                                                        SAM00020
C     W.W. GAERTNER RESEARCH INC.                                        SAM00030
C     1492 HIGH RIDGE ROAD                                               SAM00040
C     STAMFORD, CT. 06903                                                SAM00050
C     TELE (203) 322-7661                                                SAM00060
C                                                                        SAM00070
C     ********** COMPANY PRIVATE **********                             SAM00080
C                                                                        SAM00090
C     AUTHOR: W. SCHREYER                                                SAM00100
C                                                                        SAM00110
C     THIS ROUTINE CALCULATES THE FAILURE RATE OF MISCELLANEOUS PARTS    SAM00120
C                                                                        SAM00130
C     IT IS AN IMPLEMENTATION OF MIL-HDBK-217B SECTION 2.15              SAM00140
C                                                                        SAM00150
C     ALL DATA IS RECIEVED IN COMMON FROM THE CALLER                     SAM00160
C                                                                        SAM00170
C     THE FAILURE RATE IS RETURNED VIA COMMON                            SAM00180
C                                                                        SAM00190
C     DECLARE TYPES                                                      SAM00200
C                                                                        SAM00210
      INTEGER GDBIN,  SPACEL, GDFIX,  GDMOB                              SAM00220
     1       ,NAVSHL, NAVUSH, AIRINH, AIRUNH                             SAM00230
     2       ,MISLON                                                     SAM00240
     3       ,PARTA,  PARING, SCREEN                                     SAM00250
C                                                                        SAM00260
```

Figure II.3.2-24. Sample source code file containing automatically generated line sequence numbers produced on a typical commercial time sharing computer.

BEST AVAILABLE COPY

BEST AVAILABLE COPY

```
SAMPLE2  DATA      P      ID=GAERTNER          12/14/76    09.51.26          PAGE 1
         VP/CSS    ---          NATIONAL CSS, INC. (STAMFORD DATA CENTER)

 67.772   6.114  26.738   5.401  91.762   1.962  85.912  97.815  13.682BS100010
  1.754  87.389   8.546  64.769  11.706  87.312  18.518  25.301  85.146BS100020
 83.169  32.702  47.688  91.808  21.662   3.697  27.227  30.085  35.467BS100030
 42.035  33.011  19.752  21.411  59.698  11.491  12.662  72.554  21.364BS100040
 75.197  58.908  76.672  29.861  89.117  65.951  93.652  68.350  67.228BS100050
 88.224  24.289  51.714  91.687  84.696  82.991  35.682  67.175  81.911BS100060
 86.893  94.156  22.901  79.996  73.872  23.268  74.758  39.132  61.973BS100070
 19.649  60.140  83.998  62.734  20.419  57.910  63.687  60.930  92.400BS100080
  6.030   4.580  73.210  98.035  29.341  93.703  98.147  45.554  90.005BS100090
 30.040  70.194  50.807  73.098  81.321  30.045  48.379  19.866  83.783BS100100
 23.903  89.376  21.128  22.384  44.152  63.452  83.346  29.009  23.938BS100110
 82.547  79.842  36.122  98.158  63.849  99.674  23.402  43.347  49.467BS100120
  6.674  94.841   9.984   0.331  21.133  23.814  52.689   1.809  36.648BS100130
  3.613  91.842  18.534  94.627  40.956  84.094  35.964  58.939  29.958BS100140
 49.296  26.151  13.240  44.081  45.327  75.229  43.433  83.538  10.328BS100150
 10.126  67.803  15.679  83.850  61.991  17.293  45.939  19.398   3.838BS100160
 48.450  56.156   0.886  99.916  91.517  49.860  75.505   4.288  46.187BS100170
 38.527  15.480  46.138  37.508   9.804  21.254  39.289  44.450  13.102BS100180
 76.560  53.445  13.627   0.760  81.914  84.650  70.670  62.173  37.003BS100190
 62.466  41.764  88.394  54.486  31.368  97.835   4.702  47.696  43.860BS100200
 33.896   8.637  46.757   2.810  96.050  51.008  41.596  90.503  68.649BS100210
 97.369  66.371  21.909  34.105   7.457  37.798  54.675  17.864  70.110BS100220
 54.388  28.339  31.031  31.145   7.592  65.247  23.155  51.707   1.845BS100230
 45.713  57.667  34.588  98.529  19.871  22.476  56.018  33.821  98.768BS100240
 88.221  40.412  48.478  27.164  26.681  15.604  53.498  80.552   1.834BS100250
```

Figure II.3.2-25. Addition of data to file shown in Figure II.3.2-23 to show regeneration of line sequence numbers on a typical commercial time sharing computer.

The typical Government computer editor will not automatically regenerate line sequence numbers unless the file is stored without them in which case an optional extension to the edit directive must be included. It is also possible to request that the file be resequenced while it is being edited. Figure II.3.2-26 illustrates how the RESEQ command generates new line sequence numbers. Notice that the line increment value has been changed from 10 to 30.

```
COMMAND- ATTACH,S,SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO.= 004
COMMAND- EDITOR
..E,S,S

..L,A
100=    0.082    0.412    1.730    6.674   24.472
110=   86.765    0.341   21.157   23.878   52.854
120=    2.221   37.637    5.838   96.291   25.208
..RESEQ,30,30

..L,A

30=    0.082    0.412    1.730    6.674   24.472
60=   86.765    0.341   21.157   23.878   52.854
90=    2.221   37.637    5.838   96.291   25.208
..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLEDATA,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLE DATA
 CT CY= 005    0000128 WORDS.:
COMMAND-
```

Figure II.3.2-26. Regeneration of line sequence numbers using a typical Government computer.

SPS V-3.1.1.d.5.c Add Lines Without Resequencing

The capability would be automatically available to the in-house computer user. Since it is necessary to issue a command to sequence or resequence the file, then not issuing the command would not regenerate the line sequence numbers.

Employing the resources of a commercial time sharing house the ability to add lines of data to a file without regenerating line sequence numbers is optional. Upon entering the edit mode, the user merely issues the instruction SER (NO). When the file is retained the new information is included without line sequence numbers. Notice that in Figure II.3.2-27 the second line has no sequence number. This was accomplished by using the above method which is in Figure II.3.2-28.

Since the Government computer's editor used operates with the line sequence number as a pointer to the file, it is not possible to add lines of data to a file without regenerating the line sequence numbers. It should be pointed out that it would be possible to delete lines of data without resequencing under specific conditions.

SPS V-3.1.1.d.6 Copying Data Between Libraries

Copying one or more units of data from one library data file to another library data file either within a single library or between libraries can be accomplished by various methods on most time sharing systems. The following demonstrations will show how it could be accomplished in the three environments most commonly encountered by the small contractor. It will be assumed that there are two files involved in the demonstration although the actual number of files is not limited. It will also be assumed that the files are residing on random access storage. In the case of multiple libraries, one of the files may be archived on other media but the assumption is that appropriate premanipulation of the file onto a random access device has been done. Data, as opposed to source code files, will be handled in the examples. Each demonstration will show how line 3 of data file one will be copied into data file two as the fourth line. Data file one will not be altered.

Figure II.3.2-29 contains the instructions necessary to accomplish the above task on a typical in-house computer of a small contractor. The editor is invoked and file one is accessed by exercising the edit read command. This is different from the edit back command previously mentioned in that no output file is defined. The file is read and the third line is

```
SAMPLE3  DATA      P      ID=GAERTNER          12/14/76    09.51.49          PAGE 1
         VP/CSS    ---          NATIONAL CSS, INC. (STAMFORD DATA CENTER)

 67.772   6.114  26.738   5.401  91.762   1.962  85.912  97.815  13.682BS100010
 55.555   3.023
  1.754  87.389   8.546  64.769  11.706  87.312  16.518  25.301  85.146BS100020
 83.169  32.702  47.688  91.808  21.662   3.697  27.227  30.085  35.467BS100030
 42.035  33.011  19.752  21.411  50.698  11.491  12.662  72.554  21.364BS100040
 75.197  58.908  76.672  29.861  89.117  65.951  93.652  68.350  67.228BS100050
 88.224  24.289  51.714  91.687  84.696  82.991  35.682  67.175  81.911BS100060
 86.893  84.156  22.901  79.996  73.872  23.268  74.758  39.132  61.973BS100070
 19.649  60.140  83.998  62.734  20.419  57.910  63.687  60.930  92.400BS100080
  6.030   4.580  73.210  98.038  29.341  93.703  98.147  45.554  90.005BS100090
 30.040  70.194  50.807  73.098  81.321  30.045  48.379  19.866  83.783BS100100
 23.903  89.376  21.128  22.384  44.152  63.452  83.346  29.009  23.938BS100110
 82.547  79.842  36.122  96.158  63.849  99.674  23.402  43.347  49.467BS100120
  6.674  94.841   8.984   0.331  21.133  23.814  52.689   1.809  36.648BS100130
  3.613  91.842  18.534  84.627  40.956  84.094  35.964  58.939  29.958BS100140
 49.296  26.151  13.240  44.081  45.327  75.229  43.433  83.538  10.328BS100150
 10.126  67.803  15.679  83.850  61.991  17.293  45.839  19.398   3.838BS100160
 48.450  56.156   0.886  99.916  91.517  49.860  75.505   4.288  46.187BS100170
 38.527  15.480  46.139  37.508   9.804  21.254  39.289  44.450  13.102BS100180
 78.560  53.445  13.627   0.760  81.914  84.650  70.670  62.173  37.003BS100190
 62.466  41.764  88.394  54.486  31.368  97.835   4.702  47.696  43.860BS100200
```

Figure II.3.2-27. Listing of file from a commercial time sharing computer that shows addition of data without regeneration of line sequence numbers.

BEST AVAILABLE COPY

```
13.26.42 >EDIT SAMPLE DATA
EDIT:
>SER (NO)
>N 2
   1.754  87.389   8.546  64.769  11.706  87.312  18.518  25.301  85.146
BS100020
>I  55.555   3.023
>T#P 3
 67.772   6.114  26.738   5.401  91.762   1.962  85.912  97.815  13.682
B100010
 55.555   3.023
  1.754  87.389   8.546  64.769  11.706  87.312  18.518  25.301  85.146
B100020
>FILE SAMPLE3 DATA
FILING: SAMPLE3 DATA P

13.28.57 >
```

Figure II.3.2-28. Demonstration of command on commercial time sharing system to add a line of data to an existing unit without regeneration of line sequence numbers.

```
.R E
*ERSAMPLE.DA1$R$3A$V$$
  18.246   41.567   85.188   37.022   55.437
*S$$
*B/K$$
*EBSAMPLE.DAT$R$$
*4A$V$U$$
  84.626   80.885   23.679   14.110   71.548
*B/L$$

   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  18.246   41.567   85.188   37.022   55.437
  84.626   80.885   23.679   14.110   71.548
*EX$$
```

Figure II.3.2-29. Copying one unit of data from one library data file to another library data file on a typical small contractor's in-house computer.

listed. Next the save (S) command is issued with the default of one line to save. Then the pointer is placed at the beginning of the edit buffer and the buffer is cleared with the /K command. Since data file two is to be changed, the edit back command is utilized to access it. The pointer is advanced to the fourth line and the unsave command is issued. The unsave command places the entire save buffer contents, which contains line 3 from file one, before the present location of the pointer. The file is then listed showing the new line incorporated into the file.

The commercial time sharing system has greater capabilities and need not read the entire contents of file one to include line 3 of it in file two. Figure II.3.2-30 shows how the copy is accomplished. Data file two is edited and the pointer is advanced to the third line. Next the GET command is employed to retrieve line 3 from the file SAMPLE DATA1 which is automatically verified. The file is printed showing the included line.

```
13.45.47 >EDIT SAMPLE DATA
EDIT:
>N 3
   2.221  37.637   5.838  96.291  25.208
>GET SAMPLE DATA1 3 3
  18.246  41.567  85.188  37.022  55.437
>TOP 9
   0.082   0.412   1.730   6.674  24.472
  86.765   0.341  21.157  23.878  52.854
  18.246  41.567  85.188  37.022  55.437
   2.221  37.637   5.838  96.291  25.208
  84.686  80.885  23.679  14.110  71.548
>FILE

13.47.32 >
```

Figure II.3.2-30. Copying one unit of data from one library data file to another library data file utilizing a sophisticated commercial time sharing system editor.

The task may be accomplished with the typical Government computer but with the greatest degree of difficulty of the three systems. Figure II.3.2-31 demonstrates the method used. After accessing both files and invoking the editor, data file two is edited. The first three lines of the file are listed, then saved on file X. The line sequence numbers are removed with the N option. The M suffix is entered to leave the file positioned so other lines may be merged into the file. The numbers 100 and 120 give the save command the beginning and ending line numbers to save. Next, file one is edited and its third line, number 120, is saved onto file X in the same manner. File two is then rewound so it may be read again via the editor. Now lines four and five, numbers 130 and 140 of file two, are saved on file X. A peculiarity of the merge option of the save command, namely the inclusion of end of record mark, must now be removed from the file X. File X is edited and the EOR marks located, then deleted. A new file, Y, is then saved as the final desired result.

```
COMMAND- ATTACH,S1,SAMPLEDATA1,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- ATTACH,S2,SAMPLEDATA2,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- EDITOR
..E,S2,S

..L,100,120
100=    0.427    1.685    6.262   22.412   78.113
110=   66.969   98.796   90.053   51.151   96.434
120=   18.246   41.567   85.188   37.022   55.437
..S,X,N,M,100,120

..E,S1,S

..L,120

120=    2.221   37.637    5.838   96.291   25.208
..S,X,N,M,120

..REWIND,S2

..E,S2,S

..L,130,L

130=   99.429   97.641   90.982   67.123   83.901
..S,X,N,M,130,L

..E,X,S

..L,/*EOR/,A

130=*EOR
150=*EOR
..D,130

..D,150

..L,A
100=    0.427    1.685    6.262   22.412   78.113
110=   66.969   98.796   90.053   51.151   96.434
120=   18.246   41.567   85.188   37.022   55.437
140=    2.221   37.637    5.838   96.291   25.208
160=   99.429   97.641   90.982   67.123   83.901
```

Figure II.3.2-31. Copying one unit of data from one library data file to another library data file on a typical Government computer.

```
..S,Y,N

..BYE

COMMAND- CATALOG,Y,SAMPLEDATA2,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEDATA2
 CT CY= 002    00000128 WORDS.:
COMMAND-
```

Figure II.3.2-31 (continued). Copying one unit of data from one library data file to another library data file on a typical Government computer.

SPS V-3.1.2 Full Requirements

SPS V-3.1.2.a Data File Storage, Compression and Restoration.

Clear

Data file storage in a compressed mode is stated as a full requirement. The source files produced in a structured format tend to be much larger than non structured source files. Rather than being a full requirement, it is suggested that this be considered as a basic requirement. The compression technique suggested is to remove leading and trailing blanks. The question must be asked whether this is enough? If line sequence numbers are included in the lines of data, as stated as a basic requirement, how much storage would really be saved since the intervening blanks would remain with the file? The recommendation is made to consider alternative techniques such as the one employed by a commercial time sharing system. Their method is to replace strings of three or more blanks with a two byte code. Figure II.3.2-32 shows the effect this can have on a small file. The reduction ratio is 13 to 1. The greater the number of replaceable blank strings, the greater the storage saving. The file compression and restoration routines are options available to the programmer with the edit command, called PACK and UNPACK respectively.

```
13.32.47 >EDIT SAMPLE DATA OUTBY PACK
EDIT:
>FILE

13.33.04 >LISTF SAMPLE DATA*
FILENAME FILETYPE  MODE  ITEMS
SAMPLE   DATA        P     13
SAMPLE   DATAP       P      1

13.33.42 >
```

Figure II.3.2-32. Demonstration of file compression on a typical commercial time sharing computer.

The in-house minicomputer of the small contractor partially supports data compression and restoration. Tabs to specified columns suppress leading blanks. Tabs may also be contained within the lines of source code without any detrimental effect. Trailing blanks are not stored. However, when serialization is implemented this storage saving will be lost. The recommendation is made to implement a compression/restoration routine to be included as an element of a PSL operating library supplied to the small contractor.

The Government computer used does not offer a compression/restoration routine. As previously stated, storage may be saved by omitting line sequence numbers which will also remove trailing blanks from the file.

SPS V-3.1.2.b Data Maintenance

SPS V-3.1.2.b.1.a Modify a Line of Data

Modification of a line of source without having to replace the entire line has been demonstrated for files containing data. All three editors discussed have this capability for a file whether it be data or source. The examples will not be repeated here.

SPS V-3.1.2.b.1.b Scan For and Replace a Specific Data String

Scanning for and replacing a specific string of data in a single line or in every line in a unit of data in which it appears is the function of the CHANGE command in the editors. The CHANGE command was discussed for a single occurrence of

a string. Locating a specific string and changing it in an entire file can also be accomplished. The in-house minicomputer system editor accomplishes the task as shown in Figure II.3.2-33. A macro is set up to perform the operation and then the macro is executed. In the example, the string 'IPOINT' is replaced with 'JPTPL1'.

```
.R E
*EBSAMPL.FOR$RS/L$$
        A=B(IPOINT) +C(JPOINT)
        A2=B(IPOINT)*B(IPOINT)
        A3=B(IPOINT)*B(IPOINT)*B(IPOINT)
*M/GIPOINT$=CJPTPL1$/$$
*B$99EM$$
?*SRCH FAIL IN MACRO*?
*B/L$$
        A=B(JPTPL1) +C(JPOINT)
        A2=B(JPTPL1)*B(JPTPL1)
        A3=B(JPTPL1)*B(JPTPL1)*B(JPTPL1)
*EX$$
```

Figure II.3.2-33. Scanning for and replacing a specific string of data in every line in a file exploiting a typical in-house editor.

The commercial time sharing system editor accomplishes the function with a single command. Figure II.3.2-34 shows how optional extensions are appended to the CHANGE command. It should be noted that the two asterisks have the following meanings: The first represents the number of lines to search, the second represents the number of occurrences to search for in a line. Either could be set to an integer value to partially limit the CHANGE command.

```
14.23.13 >EDIT SAMPLE2 FORTRAN
EDIT:
>P 5
         A=B(IPOINT) +C(JPOINT)
        A2=B(IPOINT)*B(IPOINT)
        A3=B(IPOINT)*B(IPOINT)*B(IPOINT)
*EOF
>T
>C /IPOINT/JPTPL1/ * *
         A=B(JPTPL1) +C(JPOINT)
         A2=B(JPTPL1)*B(JPTPL1)
         A3=B(JPTPL1)*B(JPTPL1)*B(JPTPL1)
*EOF
>FILE

14.24.45 >
```

Figure II.3.2-34. Scanning for and replacing a specific string of data in every line of a file utilizing a commercial time sharing editor.

On the Government computer, the change command would also be able to accomplish the required task. Figure II.3.2-35 shows how the text replacement command is appended with 'A' to indicate all occurrences of the string in the file.

SPS V-3.1.2.b.1.c Insert a String of Data

Inserting a string of data into specific positions of every line of a file cannot be accomplished by the typical Government computer employed in the examples so far. It can be done by the other two editors.

The in-house editor will again capatilize on its macro capability. Figure II.3.2-36 contains the instructions to perform the insertion. First the file is listed, then the macro is set up. The 'A' command advances the pointer to the next line while the J command advances the pointer on a character basis. In the example, 005 is placed in columns 6 through 8 by the INSERT command.

```
COMMAND- ATTACH,S,SAMPLE2FORT,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- EDITOR
..E,S,S

..L,A

100=     A=B(IPOINT) +C(JPOINT)
110=     A2=B(IPOINT)*B(IPOINT)
120=     A3=B(IPOINT)*B(IPOINT)*B(IPOINT)
../IPOINT/=/JPTPL1/,A
  6 CHANGES

..L,A

100=     A=B(JPTPL1) +C(JPOINT)
110=     A2=B(JPTPL1)*B(JPTPL1)
120=     A3=B(JPTPL1)*B(JPTPL1)*B(JPTPL1)
..S,X,N

..BYE

COMMAND- CATALOG,X,SAMPLE2FORT,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLE2FORT
 CT CY= 002     0000128 WORDS.:
COMMAND-
```

Figure II.3.2-35. Scanning for and replacing a specific string of data in every line in a file using a typical Government computer.

```
.R E
*EBSAMPLE.DAT$RS/L$$

   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.005   80.885   23.679   14.110   71.548
*M/5JS3DSI005$VSAS/$$
*4EM$$

   0.005    0.412    1.730    6.674   24.472
  86.005    0.341   21.157   23.878   52.854
   2.005   37.637    5.838   96.291   25.208
  84.005   80.885   23.679   14.110   71.548
*EX$$
```

Figure II.3.2-36. Inserting a string of data into specific positions of every line of a file by way of a typical minicomputer editor.

The editor on a commercial time sharing system being more sophisticated has an OVERLAY command specifically designed to insert data into specific columns of a line. The argument for the OVERLAY command is a line that contains the characters to replace the existing data preceded by blanks to position the data in the proper columns. The blanks do not replace existing data. Figure II.3.2-37 shows how the OVERLAY functions.

```
15.51.41 >EDIT SAMPLE DATA
EDIT:
>P 9
    0.082    0.412    1.730    6.674   24.472
   86.765    0.341   21.157   23.878   52.854
    2.221   37.637    5.838   96.291   25.208
   84.626   80.885   23.679   14.110   71.548
*EOF
>T
> 0      005
   0.005    0.412    1.730    6.674   24.472
>A 3
   86.005    0.341   21.157   23.878   52.854
    2.005   37.637    5.838   96.291   25.208
   84.005   80.885   23.679   14.110   71.548
>FILE

15.53.49 >
```

Figure II.3.2-37. Inserting a string of data into specific positions of every line of a file utilizing a commercial time sharing system.

SPS V-3.1.2.b.2 Make Temporary Changes to a Unit of Data

Making temporary changes to a unit of data for the duration of a single job without making permanent changes is easily accomplished on the three systems available to the Small Contractor. Actually, there are alternative methods to accomplish the stated requirement depending upon the circumstances. For the examples it will be assumed that a parameter referenced on data set reference number (DSRN) one is to be the varied data. It is also assumed that the job references DSRN one via the same file name each time it executes. The assumption is also made that prior to the execution of the commands shown a

temporary data file with the appropriate changes is stored under a new name. Figure II.3.2-38 contains the instructions necessary to accomplish the task on the in-house computer. The utility program called Peripheral Interchange Program (PIP) is employed. First, a copy of the data file is made, then the temporary data file is renamed to the standard input file. Next, the program is run. Upon completion the temporary input file is renamed to its old identifier and the standard input file is returned to its proper name.

```
.R PROGRAM
        INPUT DATA

   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548

STOP--

.R PIP
*SAVFT1.DAT/R=FTN1.DAT
*FTN1.DAT/R=TEMP.DAT
*^C

.R PROGRAM
        INPUT DATA

   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901

STOP--

.R PIP
*TEMP.DAT/R=FTN1.DAT
*FTN1.DAT/R=SAVFT1.DAT
*^C
```

Figure II.3.2-38. Making temporary changes to a unit of data for the duration of a single run on the typical in-house minicomputer.

The time sharing system example will engage the ALTER command to perform the task, Figure II.3.2-39. The ALTER command operates only on the file identifier and not the file contents. It may be stated as ALTER old file to new file. As can be seen, the command sequence is almost identical.

```
16.22.36 >PROGRAM
         INPUT DATA FROM UNIT 1

    0.082    0.412    1.730    6.674   24.472
   86.765    0.341   21.157   23.878   52.854
    2.221   37.637    5.838   96.291   25.208
   84.626   80.885   23.679   14.110   71.548
STOP

16.25.37 >ALTER FILE FT01F001 P SAVE1 DATA P

16.25.56 >ALTER TEMP DATA P FILE FT01F001 P

16.26.15 >PROGRAM
         INPUT DATA FROM UNIT 1

    0.427    1.685    6.262   22.412   78.113
   66.969   98.796   90.053   51.151   96.434
   18.246   41.567   85.188   37.022   55.437
   99.429   97.641   90.982   67.123   83.901
STOP

16.29.21 >ALTER FILE FT01F001 P TEMP DATA P

16.29.40 >ALTER SAVE1 DATA P FILE FT01F001 P

16.29.59 >
```

Figure II.3.2-39. Making temporary changes to a unit of data for the duration of a single run on a commercial time sharing system.

Making use of the typical Government computer system the RENAME command will be invoked. RENAME operates similarly to ALTER. Figure II.3.2-40 contains the appropriate instructions.

```
COMMAND- PROGRAM
        INPUT DAT FROM SAMPLE DATA

   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548

  0.743 CP SECONDS EXECUTION TIME
COMMAND- ATTACH,S,SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- ATTACH,S2,SAMPLEDATA2,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- RENAME,S,SAMDATSAV

 NM ID= GAERTNER  PFN=SAMPLEDATA
 NM CY= 001     0000128 WORDS.:
 RN ID= GAERTNER  PFN=SAMDATSAV
 RN CY= 001     0000128 WORDS.:
COMMAND- RENAME,S2,SAMPLEDATA

 NM ID= GAERTNER  PFN=SAMPLEDATA2
 NM CY= 001     0000128 WORDS.:
 RN ID= GAERTNER  PFN=SAMPLEDATA
 RN CY= 001     0000128 WORDS.:
COMMAND-RETURN,S2

COMMAND- PROGRAM
        INPUT DAT FROM SAMPLE DATA

   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901

  0.747 CP SECONDS EXECUTION TIME
```

Figure II.3.2-40. Making temporary changes to a unit of data for the duration of a single run on a typical Government computer system.

```
COMMAND- ATTACH,S2,SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- RENAME,S2,SAMPLEDATA2

 NM ID= GAERTNER  PFN=SAMPLEDATA
 NM CY= 001    0000128 WORDS.:
 RN ID= GAERTNER  PFN=SAMPLEDATA2
 RN CY= 001    0000128 WORDS.:
COMMAND- RENAME,S,SAMPLEDATA

 NM ID= GAERTNER  PFN=SAMDATSAV
 NM CY= 001    0000128 WORDS.:
 RN ID= GAERTNER  PFN=SAMPLEDATA
 RN CY= 001    0000128 WORDS.:
COMMAND- RETURN,S,S2

COMMAND-
```

Figure II.3.2-40 (continued). Making temporary changes to a unit of data for the duration of a single run on a typical Government computer system.

SPS V-3.1.2.b.3 Automatically Generate Program Stubs

The capability to automatically generate program stubs for units of source code which are identified in another unit of source code being added to the PSL and are not currently stored in the PSL (i.e., have not yet been coded) is not available on the three computer systems normally available to the small contractor. Analysis of the requirement for implementation purposes leads to a two step procedure. The first step would be to identify which stubs are not yet coded. This step may not be readily apparent to the casual reader. Its function is duplicated elsewhere in the systems routines often referred to as the "link edit" process. The second step is to generate the stubs and include them in the PSL. Most likely the generated stub would have three elements. The initial element would be the entry point. The second element should be a diagnostic message that says "STUB name ENTERED", where name is replaced with the stub identifier. The inclusion of this element is important because a typographical error may cause a stub to be generated for a section of code that is already present but where the reference has been misspelled. The last element is, of course, the return to the calling routine.

It is recommended that a standard routine be developed and made available to small contractors by the Government as a

part of a PSL. It might be called ADD and accomplish the above requirement. A coroutine should also be developed called DELETE. This would enable the user to remove automatically generated stubs when it becomes time to replace them with the coded stubs.

SPS V-3.1.2.b.4 Merge Two PSL Data Files

Merging two PSL data files from two different PSL libraries into a single file can be accomplished with relative ease on the three computer systems. It is not clear if one of the data files should be appended to the other, thus destroying the first or if a third data file should be created which is a combination of the two. Both capabilities are available. However, only the second possibility will be demonstrated. It should also be noted that the actual number of data files merged is not limited to two, the only limitation being the amount of available storage. In the examples two files, A and B, will be merged to create new file C. Only one method will be shown in each example although there are alternative methods. PIP will be utilized on the small contractor's in-house computer, Figure II.3.2-41. File C.DAT is set equal to A.DAT and B.DAT, a command which PIP accepts to merge the two files. Both files shown in the example reside on random access disk. However, this is not necessary, either or both of the files could have been available from magnetic tape or any other storage device present.

When accessing the commercial time sharing system the two files must be resident on random access disk before the merge can be accomplished. The command is COMBINE as shown in Figure II.3.2-42. The structure of the command is similar to that of PIP. A major difference is that the arguments are placed on the same line as the command.

The COPY command accomplishes the merge function on the Government computer system. Figure II.3.2-43 shows how the files are accessed and merged. Since an end of record mark is included between the files on the created file, the created file is then edited to remove the EOR.

```
.R PIP
*A.DAT/L
22-NOV-76
A      .DAT     1 22-NOV-76
1316 FREE BLOCKS
*TT:=A.DAT

   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901
*B.DAT/L
22-NOV-76
B      .DAT     1 22-NOV-76
1316 FREE BLOCKS
*TT:=B.DAT

   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548
*C.DAT=A.DAT,B.DAT
*C.DAT/L
22-NOV-76
C      .DAT     2 22-NOV-76
1314 FREE BLOCKS
*TT:=C.DAT

   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901
   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548
*^C
```

Figure II.3.2-41. Merging two PSL data files from two different PSL libraries into a single file on a small contractor's in-house computer.

```
08.59.03 >LISTF * DATA
FILENAME FILETYPE MODE ITEMS
A        DATA       P     4
B        DATA       P     4

08.59.43 >PRINTF A DATA

   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901

09.02.23 >PRINTF B DATA

   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548

09.04.02 >COMBINE C DATA P A DATA P B DATA P

09.04.57 >LISTF C DATA
C        DATA       P     8

09.05.46 >PRINTF C DATA

   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901
   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548

09.08.32 >
```

Figure II.3.2-42. Merging two PSL data files from two different PSL libraries into a single data file using a commercial time sharing system available to the small contractor.

```
COMMAND- ATTACH,A,ADATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- ATTACH,B,BDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- COPYCR,A,TC

COMMAND- COPYCR,B,TC

COMMAND-EDITOR
..E,TC,S

..L,A

100=    0.427    1.685    6.262   22.412   78.113
110=   66.969   98.796   90.053   51.151   96.434
120=   18.246   41.567   85.188   37.022   55.437
130=   99.429   97.641   90.982   67.123   83.901
140=*EOR
150=    0.082    0.412    1.730    6.674   24.472
160=   86.765    0.341   21.157   23.878   52.854
170=    2.221   37.637    5.838   96.291   25.208
180=   84.626   80.885   23.679   14.110   71.548

..D,140

..S,C,N

..BYE
COMMAND- CATALOG,C,CDATA,ID=GAERTNER

 INITIAL CATALOG
 RP= 090 DAYS
 CT ID= GAERTNER  PFN=CDATA
 CT CY= 001     0000256 WORDS.:
COMMAND-
```

Figure II.3.2-43. Merging two PSL data files from two different PSL libraries into a single file as on a typical Government computer.

SPS V-3.2 Output Processing

SPS V-3.2.1 Basic Requirements

SPS V-3.2.1.a.1 Information Related to Physical Storage

Information related to the physical storage of a PSL is available on the systems used by the small contractor. The usual method employed for a PSL is to assign an entire volume for its use. Commands available in the computer systems can then be used to capture the required information.

In-house, the utility PIP offers capabilities to retrieve the information by including various switches and so-called "wild card" constructions. The amount of remaining storage available can be obtained by listing a file that does not exist, as in the first command in Figure II.3.2-44. 2302 blocks of storage are available on the disk pack. The directory space allocated is set when the disk pack is initialized. It is not shown in the example. Sufficient area is set aside to contain the entire device directory at that time. The type of storage unit must be specified in the PIP command if it is not the default system disk device. An alternative is MT0 for magnetic tape unit 0. The storage unit serial number is not available since they are not utilized by the system. Data identification is available in the form of file name and file extension. A directory listing is partially shown in the figure. It was produced by requesting a list of all files and all extensions. To obtain it alphabetically it must be put on a file then sorted. Notice that the number of blocks used, the number of free blocks and the date the listing was generated are included.

The time sharing system accessed in the following example allows concurrent multiple volume attachment to a user. Each volume is actually a portion of a disk pack or an entire magnetic tape. The disk pack volumes each have their own dynamic directories which can be identified by a letter. The STAT command, as shown in Figure II.3.2-45 allows the user to determine the amount of storage utilized and the amount remaining available in his present configuration. The present configuration may be altered dynamically by the programmer. He may engage temporary disk storage in increments of 5, 10 or 20 cylinders. If the need is for permanent allocation of a greater number of cylinders then the request must be processed overnight. The incremental value is one cylinder. The response to the STAT

```
.R PIP
*A.A/L
22-NOV-76
2302 FREE BLOCKS
*BK.LST=*.*/L
*^C

.R SORT

INPUT FILE?
*BK.LST

OUTPUT FILE?
*BK.SRT

ENTER SORT FIELD DEFINITIONS
1 10

ENTER RECORD LENGTH
25

STOP --

.R PIP
*TT:=BK.SRT
22-NOV-76
125 FILES, 2476 BLOCKS
2302 FREE BLOCKS
B51BP1.MLC    21 12-JUL-76
B51BP2.MLC    68 15-JUL-76
B51BP2.SGL   277 14-JUL-76
B51BP2.SRT   276 15-JUL-76
B51CI .DAT    21  4-AUG-76
B51CKI.FOR     4 13-JUL-76
B51CKI.OBJ    12 13-JUL-76
B51CKI.SAV    16 13-JUL-76
B51DIS.MAC     2 30-AUG-76
B51DIS.PB5     1 30-AUG-76
```

Figure II.3.2-44. Library control listings generated on an in-house computer of a small contractor.

```
11.02.16 >STAT
P-DSK(WT): 00180 USED, 00116 LEFT(OF 00296), 61% (OF 002 CYL)

11.02.48 >DISKE
 *ENTER SORT FIELD DEFINITIONS
   9 RECORDS READ ON PASS 1
P-DSK(WT): 00180 USED, 00116 LEFT(OF 00296), 61% (OF 002 CYL)

11.04.35 >PRINTF SORTLIST EXEC
 &1 &2 A         MEMO        P      4
 &1 &2 ACCTPTL2  EXEC        P     12
 &1 &2 ADDRESS   MEMO        P      4
 &1 &2 B         MEMO        P      3
 &1 &2 C         MEMO        P      4
 &1 &2 DISKE     EXEC        P      9
 &1 &2 MARCTL    EXEC        P     19
 &1 &2 MARPLN    DATA        P    386
 &1 &2 MARPLN    EXEC        P     18

11.06.13 >
```

Figure II.3.2-45. Library control listings generated on commercial time sharing system.

command gives the disk reference letter, P in this case; the mode it is being accessed in, WT for writable; the number of records used, left and the total available; the percentage of the cylinders containing data and the total number of cylinders. The directory area is fixed length and is not alterable by the user. The DISKE command produces an alphabetical directory listing. This is a stored sequence of commands that combines the LISTF command, SORT command, and STAT command to produce the desired result.

The Government computer has the capability to provide much of the information requested via the AUDIT command. The system mass storage device is shared by all users, each one retaining the amount desired. The AUDIT command operates on the entire device though, and the programmer must therefore append extensions to the AUDIT command in order to obtain information about his own files selectively. The directory listing as shown in Figure II.3.2-46 is not alphabetical. However, it may be sorted. The listing is in printer format, 132 columns, but it is shown here as it is typed on the user's terminal.

```
COMMAND- AUDIT,ID=ADEMO,AI=P


 AUDIT OF 6000 PERMANENT FILES  PARTIAL   ID                    TIME
16.28.16      07/08/76                PAGE NO.     1

 SETNAME=SYSTEM:

 OWNER        PERMANENT FILE NAME                          CYCLE  ACCOUNT
  UNIT  PRUS  CREATION  EXPIRATION  LAST ATT   LAST ALT
 ADEMO        DEMOPARTS                                      1    MGR333
 0050       2  07/08/76  10/06/76    07/08/76   07/08/76
 ADEMO        DEMOSRTD                                       1    MGR333
 0050       3  07/08/76  10/06/76    07/08/76   07/08/76
 ADEMO        DIAGNOSTICBINNY                                1    MGR333
 0053      13  07/08/76  10/06/76    07/08/76   07/08/76
 ADEMO        DIAGNOSTIC                                     1    MGR333
 0052       7  07/08/76  10/06/76    07/08/76   07/08/76
 ADEMO        FOUNDO                                         1    MGR333
 0052       5  07/08/76  10/06/76    07/08/76   07/08/76
 ADEMO        DEMOSUMD                                       1    MGR333
 0051       2  07/08/76  10/06/76    07/08/76   07/08/76
 ADEMO        DEMOABS                                        1    MGR333
 0052       5  07/08/76  10/06/76    07/08/76   07/08/76
 ADEMO        USERDIAGNOSTIC                                 1    MGR333
 0051       4  07/08/76  10/06/76    07/08/76   07/08/76

 AUDIT FINISHED
 EXIT
COMMAND-
```

Figure II.3.2-46. Library control listing generated on a Government computer system.

BEST AVAILABLE COPY

SPS V-3.2.1.b.1 Source Data Listings

Source data listings of selected units of data is assumed to mean printing of specific lines of data in a file. There is no direct facility available in-house. It is possible to exploit the editor to selectively print selected lines as shown in Figure II.3.2-47. It would be desirable to have this capability and the recommendation is made that it become an element of the PSL supplied by the Government. Its format should follow that used by the commercial time sharing system.

```
.R E
*ERSAMPLE.DAT$R$$
*2A$3L$$
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901
*15A$2L$$
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548
*EX$$
```

Figure II.3.2-47. Listing of selected units of data using a typical in-house computer.

The commercial time sharing system has available the PRINTF command to produce selective units-of-data listings. The user specifies the file to print from the beginning line number and the ending line number. An added feature of this command is that selected volumes of data may also be printed. Figure II.3.2-48 displays an example of the PRINTF command. Lines 3 through 5 of the SAMPLE DATA are printed.

```
11.34.26 >PRINTF SAMPLE DATA 3 5

   1.010    3.342   10.959   35.678   15.439
  16.659   30.709   34.320   29.542   68.369
  44.335   50.686    5.103   74.447    0.753

11.35.19 >
```

Figure II.3.2-48. Listing of selected units of data by a commercial time sharing system.

The Government computer system does not offer direct printing of selected units of data. Similarly to the in-house system files may be edited and selected lines printed as shown in Figure II.3.2-49.

```
COMMAND- ATTACH,S,SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO:.= 001
COMMAND- EDITOR
..E,S,S

..L,120,140

120=   71.531   90.235   97.627   73.649   63.250
130=   16.659   30.709   34.320   29.542   68.369
140=   44.335   50.686    5.103   74.447    0.753

..BYE

COMMAND-
```

Figure II.3.2-49. Listing of selected units of data produced by a Government computer.

SPS V-3.2.1.b.2 Listing of all the Data in a Single Library Data File

Listing of all the data in a single library data file can be accomplished by all three systems. In-house, PIP produces the listing. The output device in the example, Figure II.3.2-50, has been set to the terminal. However, it could just as easily have been assigned to the line printer. The file SAMPLE DAT is printed in its entirety.

The commercial time sharing system provides a command to produce listings of files called OFFLINE PRINT. The arguments for the command are the file identifiers. Figure II.3.2-51 shows the results of the command "OFFLINE PRINT SAMPLE DATA".

```
.R PIP
*TT:=SAMPLE.DAT

   1.010    3.342   10.959   35.678   15.439
  71.531   90.235   97.627   73.649   63.250
  16.659   30.709   34.320   29.542   68.369
  44.335   50.686    5.103   74.447    0.753
   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901
   0.082    0.412    1.730    6.674   24.472
  86.765    0.341   21.157   23.878   52.854
   2.221   37.637    5.838   96.291   25.208
  84.626   80.885   23.679   14.110   71.548
*
```

Figure II.3.2-50. Listing of an entire file on the in-house minicomputer.

```
SAMPLE   DATA      )      ID=GAERTNER          12/14/76    09.51.12          PAGE 1
         VP/CSS    ---          NATIONAL CSS, INC. (STAMFORD DATA CENTER)

 67.772   6.114  26.738   5.401  91.762   1.962  85.912  97.815  13.682RS100010
  1.754  87.384   8.546  64.769  11.706  87.312  18.518  25.301  85.146RS100020
 83.169  32.702  47.688  91.808  21.662   3.697  27.227  30.085  35.467RS100030
 42.035  33.011  19.752  21.411  50.698  11.491  12.662  72.554  21.364RS100040
 75.197  38.908  76.672  29.861  89.117  65.951  93.652  68.350  67.228RS100050
 68.224  24.289  51.714  91.687  84.696  82.991  35.682  67.175  81.911RS100060
 96.393  84.156  22.901  79.996  73.972  23.268  74.758  39.132  61.973RS100070
 19.649  60.140  83.998  62.734  20.419  57.910  63.687  60.930  92.400RS100080
  6.030   4.580  73.210  98.036  29.341  93.703  98.147  45.554  90.005RS100090
 30.040  70.194  50.807  75.098  91.321  30.045  48.379  19.866  83.783RS100100
 23.903  89.376  21.128  22.384  44.152  62.452  83.346  29.009  23.938RS100110
 42.547  79.342  36.122  98.158  63.849  99.674  23.402  43.347  49.467RS100120
  6.674  94.841   8.984   0.331  21.133  23.814  52.689   1.809  36.648RS100130
 3.613  91.842  18.534  84.627  40.956  84.094  35.964  58.939  29.958RS100140
49.296  26.151  13.240  44.081  45.327  75.229  43.433  83.538  10.328RS100150
10.126  67.803  15.679  83.850  61.991  17.293  45.839  19.398   3.539RS100160
48.450  56.156   3.886  99.916  91.517  49.860  75.505   4.288  46.187RS100170
38.527  15.480  46.139  37.508   9.804  21.254  39.289  44.450  13.102RS100180
75.560  53.445  13.627   0.760  81.914  84.650  70.670  62.173  37.003RS100190
62.406  41.764  88.394  54.486  31.368  97.835   4.702  47.696  43.860RS100200
```

Figure II.3.2-51. Listing of an entire file on a commercial time sharing system.

BEST AVAILABLE COPY

The Government computer does not offer direct programmer access to the printer. The user may direct listings to the printer by using a batch method. The command necessary is "BATCH, PRINT, LFN, GAER". This will print a file with the local file name of LFN. Figure II.3.2-52 displays a listing produced by this command.

```
1[illegible].333  [illegible]  63.737  25.624
 35.726  15.[illegible]  [illegible].351  [illegible].[illegible]23
147.[illegible]25  41.[illegible]75  2[illegible]4.836  51.759


***********          IGAER1S  //// END OF LIST ////
***********          IGAER1S  //// END OF LIST ////
```

Figure II.3.2-52. Listing an entire file on a typical Government computer.

SPS V-3.2.1.c Control Data

Control data as described in Volume V, paragraph 3.2.1.c is definitely a basic requirement of any system that is being developed. The basic attributes that identify a listing are not available in house. It would be possible to type the DATE and TIME commands to present the information on the terminal but not on the line printer. The file name is already shown on the terminal. Figure II.3.2-51 shows the example listing from the commercial time sharing system indicating that the information is already present. Also, the page number is automatically inserted and incremented. The typical Government computer has no file identification or, time and date automatically inserted in the listing. It should be noted that both the in-house computer and the Government computer produce compiler listings that do contain the date and time but not the file name. This would be of no help to the programmer applying structured methodology, since he will rarely see or even produce these listings when using a precompiler.

SPS V-3.2.2 Full Requirements (OUTPUT)

SPS V-3.2.2.a Magnetic Tape Output

This small contractor has found magnetic tape transfer of data to be the most economical method for distribution to other computer facilities. Over an eleven year period no data has been lost from the magnetic tapes while being conveyed over land or in the air utilizing both the Postal Service and commercial air carriers. Tests conducted by the National Bureau of Standards confirm our experience (see DATAMATION, March 1976, pp. 65-70 "Erasing Myths About Magnetic Media"). Copying one or more units of data onto magnetic tape may be accomplished in all of the three systems discussed as long as the unit of data is contained on the storage medium as a complete file. Since this is the case, examples for copying a file on magnetic tape will be shown in each system. Although only one file is copied, the number of files which can be copied is only limited by the amount of information the tape can hold.

The PDP-11/40 used in-house by this small contractor has the capability to copy files to magnetic tape employing the utility program PIP, Figure II.3.2-53. The output device, mag tape unit 0, MT0, is placed in front of the output file name. Since the input file resides on disk, no device identifier is entered. To show that the file has been copied, the file on tape is listed.

```
.R PIP
*MT0:SAMPLE.DAT=SAMPLE.DAT
*MT0:/L
22-NOV-76
SAMPLE.DAT     1 22-NOV-76
*
```

Figure II.3.2-53. Copying one file of data onto magnetic tape using a typical in-house minicomputer.

On the time sharing system, a program called UTILITY is available to copy files between media. UTILITY generates tapes that can be read on IBM, DEC and CDC equipment. In the example,

Figure II.3.2-54, the tape is mounted by requesting the operator to do so. UTILITY is started by typing its name. Next, commands are given to direct the copy: First the output device is specified, then the output density is selected. Next the desired transfer is commanded with DT, meaning disk to tape. UTILITY now prompts for the input file identifier and the blocking factor. After the file is transferred, UTILITY prompts for the next operation, END in this case, signifying that the user is finished. Either more files could have been transferred or the file could have been checked for proper copying.

```
12.45.08 >MOUNT TAPE X3476 AS 282 RINGIN 9TRK-LO

12.46.12 >DEV 282 ATTACHED
>T REW

12.51.35 >UTILITY
REQUEST: OUT=TAP2
REQUEST: DEN,OUT,2
REQUEST: DT
INPUT FILE?
SMPLE DATA
BLOCKING FACTOR
1
REQUEST: END

12.52.16 >
```

Figure II.3.2-54. Copying one file of data onto magnetic tape using a commercial time sharing system.

The Government computer does not allow a magnetic tape unit to be directly attached to the user's machine. Copying of files to tape must therefore be performed indirectly through batch mode. A sample job stream to accomplish the copy is contained in Figure II.3.2-55. Notice that a FORTRAN program is included in the stream to perform the copy operation since there is no system utility command. The tape generated can be read by IBM, DEC or CDC equipment.

```
COPYFIL,NT1.
COMMENT.(AAA-BBB,NNNNNT),JONES
ATTACH,TAPE1,SAMPLEDATA,ID=GAERTNER.
REQUEST,TAPE2,NT,S. BIN/1122,REEL/4031, RINGIN.
FTN.
LGO.
*EOR
        PROGRAM CPYFL(TAPE1,TAPE2,OUTPUT)
        REAL A(8)
2       FORMAT(8A10)
1       READ(1,2)A
       IF(EOF(1).NE.0) STOP
        BUFFER OUT (2,0) (A(1),A(8))
        WAIT=UNIT(2)
       GOTO 1
       END
```

Figure II.3.2-55. Copying one file of data onto magnetic tape using a typical Government computer.

SPS V-3.2.2.b Punched Card Output

Punched card output is suggested as a full requirement. The small contractor has the capability to produce punched cards readily via a time share system, but in the majority of installations an in-house minicomputer system do not contain a card-punch peripheral. If it did, Figure II.3.2-56 illustrates how a file would be punched. The method is almost identical to the one which applies to the production of magnetic tape, using the PIP program with an output-device identifier of CP, the card punch.

```
.R PIP
*CP:SAMPLE.DAT=SAMPLE.DAT
*↑C
```

Figure II.3.2-56. Punching a file to cards on an in-house minicomputer.

On the time sharing system, two methods are available to produce card decks of files. One method would be to exploit UTILITY, the alternative is to issue the OFFLINE PUNCH command. The second method is presented in Figure II.3.2-57. This mode is limited to disk-to-card transfers, while UTILITY can also punch from magnetic tape.

```
13.08.41 >OFFLINE PUNCH SAMPLE DATA

13.08.59 >
```

Figure II.3.2-57. Punching a file to cards on a commercial time sharing system.

In both of the above systems, the card punch is a virtual portion of the user's machine as was the tape drive. However, the Government computer used in this study leaves no doubt to the fact that the central operating system is the only one capable of punching cards or accessing magnetic tape. The programmer must specifically direct any data he wishes punched to the central system for the operation. Figure II.3.2-58 shows how the user would punch a deck of cards.

```
COMMAND- ATTACH,S,B55SAMPLEDATA,ID=GAERTNER

 PF CYCLE NO. = 001
COMMAND- EDITOR
..E,S,S

..L,A,S

   0.381    1.657    6.509   24.143   86.271
   0.341   25.607   50.576   72.991   82.768
  39.686   93.204    2.049   73.456   22.296
  72.675   35.383   58.224   30.897   61.368
   0.381    1.657    6.509   24.143   86.271
   0.341   25.607   50.576   72.991   82.768
  39.686   93.204    2.049   73.456   22.296
  72.675   35.383   58.224   30.897   61.368

..BYE

COMMAND- BATCH,S,PUNCH,GAER

 FILE NAME-IGAERA0 ,DISP-PUNCH  ,ID-**
COMMAND-
```

Figure II.3.2-58. Punching a file to cards on a typical Government computer.

SPS V-3.2.2.c Management Control Listings

Clear

SPS V-3.2.2.d Programmer Directory and Unit Listings

Programmer directory and unit listings can be generated on all three systems. However, the degree of automation varies dramatically. Each file on the system is identified by a name with a length of six characters on the in-house system, eight characters on the time sharing system and 40 characters on the Government system. The first two also require an extension or file type identifier. It specifies the use of the file. The third system allows a unique identifier as a second required parameter.

This small contractor has found it advantageous to develop standard program identification techniques that associate the program with the programmer and the contract. In-house and on the commercial time sharing system the first character of a file-name identifies a programmer. Due to the small number of programmers involved, a single-character identifier usually suffices. The second and third characters assign a project code to the file. The last three or five characters identify the program. On the Government system, the unique ID associates the file with a particular contract, with nine characters allowed. The forty character file name identifies the programmer, program and type of file.

In-house, the utility program DIR generates a list of all programs by a particular programmer. DIR was written by this contractor specifically for his system. However, the program is relatively simple and could either be developed by all small contractors or produced in a general form and distributed as an element of a PSL by the Government. DIR duplicates the list function of PIP but allows more global parameters than PIP does. Entering a programmer project code identifier to DIR generates a list of names of all programs written by a programmer. It is not possible to automatically print listings of the files at this time, but the idea has been considered. PIP must currently be used to print the files. By including more than one file identifier on a line, pseudo-automatic operation is achieved. Figure II.3.2-59 shows how DIR displays all data files generated by the programmer with code B, on project code 55, are then printed with one command to PIP.

```
.R DIR

DIR
B55.DAT

B55MIN.DAT      1  19-NOV-76
B55MAX.DAT      1  19-NOV-76
B55AVG.DAT      1  19-NOV-76


DIR
^C

.R PIP
*TT:=B55MIN.DAT,B55AVG.DAT,B55MAX.DAT
 1.00  -7.00  63.00
38.00 -33.00 118.00
73.00  -1.00 176.00
*
```

Figure II.3.2-59. Programmer directory and file listings generated on a small contractor's in-house minicomputer.

Features of the sophisticated time sharing system already allow global extensions within the LISTF command to selectively list the file names created by a programmer on a specific project. Figure II.3.2-60 contains an example of LISTF. Notice that an extension was used on the command that creates a file. The file is executed to automatically print the files on the terminal with the PRINTF command. An alternative would have been the OFFLINE PRINT command to direct the files to the line printer.

On the Government computer used there are no global options to the AUDIT command that give the versatility of the previous example. The most direct method to accomplish the task would be to direct the AUDIT output to a local file which could then be edited. The LIST command could then be issued to find the identifying strings in the particular columns where they occur. Then, to print the files each would have to be attached and sent through batch mode to the line printer. Figure II.3.2-61 shows how the above is accomplished.

```
13.29.34 >LISTF B55* DATA (E NOITEM)

13.29.57 >P LISTF EXEC

 &1 &2 B55RANDU DATA        P
 &1 &2 B55MAXIM DATA        P
 &1 &2 B55MINIM DATA        P

13.31.12 >LISTF PRINTF

13.31.38 PRINTF B55RANDU DATA P

 29.014   57.732   31.001    7.887

13.32.03 PRINTF B55MAXIM DATA P

100.000 100.000 100.000 100.000

13.32.54 PRINTF B55MINIM DATA P

  1.000    1.000    1.000    1.000

13.33.21 >
```

Figure II.3.2-60. Programmer directory and file listings generated on a commercial time sharing system.

```
COMMAND- AUDIT,LFN=LIST,AI=P,ID=GAERTNER

   AUDIT FINISHED
  EXIT
COMMAND- EDITOR
..E,LIST,S

..L,/B55/,A,(15,18)

  130=   GAERTNER     B55MAXIMUMDATA                                          1
 MGX111   0050        2  07/08/76  10/06/76     11/22/76    10/22/76
  170=   GAERTNER     B55MINIMUMDATA                                          1
 MGX111   0050        2  07/08/76  10/06/76     11/22/76    10/22/76
  260=   GAERTNER     B55AVERAGEDATA                                          1
 MGX111   0050        2  07/08/76  10/06/76     11/22/76    10/15/76

..BYE
COMMAND- ATTACH,X,B55MINIMUMDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- BATCH,X,PRINT,GAER

 FILE NAME-IGAER17 ,DISP-PRINT  ,ID-**
COMMAND- ATTACH,X1,B55AVERAGEDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- BATCH,X1,PRINT,GAER

 FILE NAME-IGAER21 ,DISP-PRINT  ,ID-**
COMMAND- ATTACH,X2,B55MAXIMUMDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- BATCH,X2,PRINT,GAER

 FILE NAME-IGAER22 ,DISP-PRINT  ,ID-**
COMMAND-
```

Figure II.3.2-61. Programmer directory and file listing generated on a typical Government computer.

SPS V-3.2.2.e Automatic Indentation of Source Code

The capability to automatically indent source code generated from PSL source data files is not available on any of the three systems most often used by the small contractor. It is recommended that a program for automatic indentation be developed and supplied by the Government to small contractors. Careful design techniques must be followed when trying to produce such a program, especially when FORTRAN is addressed, which does not allow complete indentation. The C of a comment card must be in column one, the statement numbers must appear in columns one through five, the continuation character must fall in column six. Another problem occurs if a line, when indented, needs to be extended into a continuation line. The continuation line could be automatically inserted but the program would also have to ensure that the number of continuation lines did not exceed the limit allowed by the compiler.

SPS V-3.2.2.f Data Scanning

Scanning for a specific string of data in every line of every unit in a PSL and listing the names of the units in which it appears can be accomplished on each of the systems discussed. The method is not a direct implementation of the stated task, but the desired results can be achieved. It is recommended that an exact interpretation be developed as an element of a set of programming support functions that would be supplied by the Government to small contractors. Each of the systems would store the PSL units as separate files. Each file could then be edited and a LOCATE for the specific string over the entire file be executed. The list of output would show each file name followed by either a list of the occurrences of the string in the file or an error message indicating that the string was not found. An added advantage of this method is that the usage of the string will be displayed. Each of the systems allows two procedures to perform the task, one requiring more programmer intervention than the other. The first method is for the user to enter the edit and locate commands from the terminal and have the commands execute as entered. This method is shown in Figure II.3.2-62 on the in-house system. The string COEFF is searched for in three files.

```
.R E
*ERB55CMA.FOR$R$M/GCOEFF$V$/$99EM$$
        CODE  =ALPHA * BETA *COEFF
        SAVCOF =COEFF
        COEFF =SAVCOF
?*SRCH FAIL IN MACRO*?
*B/K$$
*ERB55CMB.FOR$R$B$99EM$$
?*SRCH FAIL IN MACRO*?
*B/K$$
*ERB55MAT.FOR$R$B$99EM$$
C                MATCH COEFFICIENT TO ARRAY
        IF(COEFF .EQ. ARRAY(I,J) ) GOTO 100
C               COEFFICIENT MATCHED TO ARRAY
?*SRCH FAIL IN MACRO*?
*↑C
```

Figure II.3.2-62. Scanning for a specific string of data in every line of every unit in a PSL and listing the names of the units in which it appears using a typical in-house minicomputer.

The second alternative is to set up a series of commands to perform the search. This method is shown for the commercial time sharing system. First, the sequence of commands is created as a file, as exemplified in Figure II.3.2-63.

```
09.23.31 >P SCAN EXEC

EXEC STKDCMND
EDIT B55COM19 FORTRAN
EXEC STKDCMND
EDIT B55COMPB FORTRAN
EXEC STKDCMND
EDIT B55MATCH FORTRAN
```

Figure II.3.2-63. Example of a stored sequence of commands on a commercial time sharing system to scan for a specific string of data in every line of every unit of data in a PSL and list the unit names in which it appears.

The line "EXEC STKDCMND" performs the commands in file STKDCMND EXEC. This file contains the search commands which are identical for each file. Figure II.3.2-64 lists the STKDCMND EXEC file.

```
09.31.42 >P STKDCMND EXEC

&BEGSTACK
LOCATE /COEFF/ * *
QUIT
&ENDSTACK
```

Figure II.3.2-64. Listing of file STKDCMND EXEC

The above two files offer a high degree of flexibility. The SCAN EXEC file can be updated as the library expands to include all new entries into the library. The file STKDCMND EXEC could be changed to store any editor command or include any system or exec commands. The execution of the search is shown in Figure II.3.2-65.

```
09.53.52 >SCAN

EDIT B55COMPA FORTRAN
EDIT:
        CODE  =ALPHA *BETA *COEFF
        SAVCOF=COEFF
        COEFF =SAVCOF
EOF
09.54.51 EDIT B55COMPB FORTRAN
EDIT:
EOF
09.54.58 EDIT B55MATCH FORTRAN
EDIT:
C               MATCH COEFFICIENT TO ARRAY
      IF( COEFF .EQ.ARRAY(I,J) ) GOTO 100
C               COEFFICIENT MATCHED TO ARRAY
EOF

09.56.03 >
```

Figure II.3.2-65. Scanning for a specific string of data in every line of every unit in a PSL and listing the names of the units in which it appears using a stored sequence of commands on a commercial time sharing system.

The interactive terminal technique will be demonstrated for the Government computer system. The sequence of commands is to attach and scan one file after another. Notice that the attach command can be executed from within the edit environment, Figure II.3.2-66.

```
COMMAND- ATTACH,X,B55COMPUTEAFORTRAN,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- EDITOR
..E,X,S
..L,/COEFF/,A

 190=              CODE  =ALPHA *BETA *COEFF
 380=              SAVCOF=COEFF
1520=              COEFF =SAVCOF

..ATTACH,XX,B55COMPUTBFORTRAN,ID=GAERTNER

 PF CYCLE NO.= 001
..E,XX,S
..L,/COEFF/,A

..ATTACH,XE,B55MATCHFORTRAN,ID=GAERTNER

 PF CYCLE NO.= 001
..L,/COEFF/,A

 350=C                    MATCH COEFFICIENT TO ARRAY
 470=               IF( COEFF .EQ.ARRAY(I,J) ) GOTO 100
 610=C                    COEFFICIENT MATCHED TO ARRAY

..
```

Figure II.3.2-66. Scanning for a specific string of data in every line of every unit in a PSL and listing the names of the units in which it appears, on a Government computer.

SPS V-3.3 Programming Language Support

SPS V-3.3.1 Basic

SPS V-3.3.1.a.1 Syntax Checking of Source Code

Syntax checking of source code is done by all compilers. However, the degree to which the compilers check for syntax errors varies. Each computer manufacturer has developed his

own compiler or compilers for the popular higher-level languages and additional compilers have been developed for specific purposes by non-manufacturers. This small contractor's experiences with various compilers has been broad and therefore we offer a few insights as to their differences: For example, the FORTRAN standard specifies that a variable name is limited to six characters. Both CDC and DEC allow the use of seven character variable names. The WATFIV compiler has the capability to determine if a variable is undefined before it is used while the IBM FORTRAN IV levels G and H do not have this capability. There are few compilers that strictly adhere to the ANS standard. Each compiler writer decides that certain extensions would be desirable for either convenience or to offer specific options that his computer has available. If any of the language standards is updated, as has been done with COBOL, it is recommended that a knowledgeable structured programming representative be included in the effort. While syntax validation will remain a task of the compiler or precompiler, the use of extensions within various compilers has generally liberalized the syntax requirements of many languages.

SPS V-3.3.1.a.2 Compilation of Source Code and Storing the Object Module

Compilation of source code and the storage of the object module is accomplished by many compilers. All three of the systems used as examples here are capable of doing this. An exception to this are the "compile and go" compilers, which do not produce object modules that can be saved. They were developed for a different purpose, namely for instructional environments where a compiler that utilizes relatively small amounts of compile time at the expense of execution time is required due to the large number of compiles needed in the particular situation. Where it was found that the number of program runs that were made after the code was debugged was very small with respect to the number of compiles. A typical example of such a compiler is WATFIV.

The in-house computer often used by the small contractor has the capability to compile and save the object module. In the example shown in Figure II.3.2-67, the FORTRAN compiler is invoked to compile the source code contained in file BESSAM.FOR. Since the input and output file extensions are not specified the compiler automatically assumes the input extension to be FOR and the output extension to be OBJ. The two files are then listed via PIP.

```
.R F
*B55SAM=B55SAM
*↑C

.R PIP
*B55SAM.*/L
21-SEP-76
B55SAM.FOR     1 16-SEP-76
B55SAM.OBJ     3 21-SEP-76
2 FILES, 4 BLOCKS
3127 FREE BLOCKS
*
```

Figure II.3.2-67. Compiling of source code and storing of the resulting object module by an in-house computer.

The time sharing system also has the ability to compile a source file and store the object module. To invoke the compiler the programmer types the language name followed by the name of the file to be compiled, optionally including various compile time directives. Options of interest to the programmer using structured techniques are those which perform automatic validity checking of subscripts during execution, invoke syntax and allocation checking and allow conditional compilation of statements. Figure II.3.2-68 contains an example compile of a FORTRAN program stored under the file name B55SAMPL. After the compilation, the LISTF command is issued to show the object module created which has the same file name but with the file type of TEXT.

```
15.37.02 >FORTRAN B55SAMPL

15.38.14 >LISTF B55SAMPL *
FILENAME FILETYPE MODE ITEMS
B55SAMPL FORTRAN     P    313
B55SAMPL TEXT        P     27

15.29.43 >
```

Figure II.3.2-68. Compiling a source file and automatically storing the object module on a commercial time sharing system.

The typical Government computer has the capability to compile source code and store the resultant object code. The compiler is initiated by typing FTN and identifying the input file to be compiled. The output is put on the local file called LGO which the user must save with the CATALOG command as seen in Figure II.3.2-69.

```
COMMAND- ATTACH,X,B55SAMPLEFORTRAN,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- FTN,I=X

 0.124 CP SECONDS COMPILATION TIME
COMMAND- CATALOG,LGO,B55SAMPLETEXT,ID=GAERTNER

 INITIAL CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER PFN=B55SAMPLETEXT
 CT CY= 001     0000256 WORDS.:
COMMAND-
```

Figure II.3.2-69. Compiling a source file and storing the resultant object module on a typical Government computer.

SPS V-3.3.1.b Load Module Generation

Load modules can be generated on practically all time sharing systems. The actual methods vary as do the command names to perform the operation. On the in-house computer a linker is employed to combine one or more object modules with the required system and language library routines into a memory image module. The resultant module is automatically saved on random-access disk storage. The program is then ready to run. The linker is initiated by the command R LINK. The instruction to the linker identifies the output file, the input file or files, any libraries that should be searched to satisfy external references and any additional optional parameters. Figure II.3.2-70 contains the instructions to create a load module, and run the program after listing the file created.

```
.R LINK
*B55SAM=B55SAM/F
*↑C

.R PIP
*B55SAM.SAV/L
17-SEP-76
B55SAM.SAV    9  17-SEP-76
*↑C

.R B55SAM

        INPUT DATA

   1.010    3.342   10.959   35.678   15.439
  71.531   90.235   97.627   73.649   63.250
  16.659   30.709   34.320   29.542   68.369
  44.335   50.686    5.103   74.447    0.753
.
```

Figure II.3.2-70. Creating a load module that is saved on disk and executing it on a typical in-house computer.

A load module is generated on the commercial time sharing system in the manner shown in Figure II.3.2-71. The file or files are loaded into core with the LOAD command. Various options are available to specify map printing, core disposition before loading begins, what libraries to search for unsatisfied externals, and the order in which to search the libraries. If the programmer wishes to specify more file names than can be typed on one line then the USE command may be issued with the additional file names. Upon successful loading, the GENMOD command is entered to create a core-image file on disk. To execute the file stored on disk, the user types LOADMOD followed by the file name to load. Then the START command is issued.

The Government computer permits generation of a load module in a different manner. If multiple object files are to be included they must first combine into one file. This file may then be loaded for the module generation. The command

```
15.13.33 >LOAD B55SAMPL

15.14.01 >GENMOD B55SAMPL

15.14.41 >LISTF B55SAMPL MODULE
B55SAMPL MODULE    P    2

15.16.05 >LOADMOD B55SAMPL

15.16.43 >START

        INPUT DATA

  0.427    1.685    6.262   22.412   78.113
 66.969   98.796   90.053   51.151   96.434
 18.246   41.567   85.188   37.022   55.437
 99.429   97.641   90.982   67.123   83.901
```

Figure II.3.2-71. Creating a load module that is saved on disk and subsequently recalled for execution on a commercial time sharing system.

to generate the module is XEQ, Figure II.3.2-72. To save the load module the programmer must have previously issued a REQUEST command specifying that the file created is to reside on a permanent device as opposed to a temporary device which is the default. Various options are available with the XEQ command similar to those of the commercial time sharing system.

SPS V-3.3.2 Full Requirements

SPS V-3.3.2.a Top Down Structured Programming Support

SPS V-3.3.2.a.1 Flagging Explicit Branches

Flagging of all explicit branches in source code listing is not a capability provided on any of the systems so far discussed as being available to the small contractor. It is recommended that a program be developed to scan a source file for all explicit branches. The program would be included as an element of a functional PSL package provided by the Government to small contractors. It should have the capability to identify the explicit branches and list them by source line number and also display them.

```
COMMAND- REQUEST,SAMPLE,*PF

COMMAND- ATTACH,X,B55SAMPLETEXT,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- XEQ

OPTION=LOAD=X
OPTION=NOGO
COMMAND-CATALOG,SAMPLE,SAMPLE,ID=GAERTNER

 INITIAL CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLE
 CT CY= 001     00000128 WORDS.:
COMMAND- REWIND,SAMPLE

COMMAND- XEQ,SAMPLE

        INPUT DATA

   0.427    1.685    6.262   22.412   78.113
  66.969   98.796   90.053   51.151   96.434
  18.246   41.567   85.188   37.022   55.437
  99.429   97.641   90.982   67.123   83.901

COMMAND-
```

Figure II.3.2-72. Creating a load module which is saved on disk and later executed on a typical Government computer.

SPS V-3.3.2.a.2 Flagging Source Code Units That Exceed a Maximum Size

Flagging of program language source code units that exceed a maximum size to be defined by the user is not provided as a function on any of the systems the small contractor presently has at his disposal. The small contractor is able to determine the amount of storage required by each unit as previously shown by PIP, LISTF or AUDIT. Each is able to create a file of the library names and storage actually utilized. This file would be compared with a programmer-generated table that contains a file maximum length field to produce the desired flagging. An alternative method would be to access each file and compare its line count to the line count in the limit file. An additional feature might be to generate a list of files that are not yet defined on the storage device but are defined in the limit table and a list of files that are present in the library but not included in the table. Various percentages could be determined to show the estimated versus the actual amount of code.

SPS V-3.3.2.a.3 Flagging Source Lines That Contain Multiple Statements

Flagging any lines of program source code that contain more than one structured source statement is not available to the small contractor at present. This full requirement could be included in a functional PSL supplied to the small contractor by the Government. The recommendation is to do so. When developing this program care must be exercised in determining that statements allowed by the language are structured properly. For example, FORTRAN allows the programmer to place the GOTO portion of an IF statement on the same source code line as the IF. In Structured FORTRAN the GOTO must be placed on a continuation line.

SPS V-3.4 Library System Maintenance

SPS V-3.4.1 Basic Requirements

SPS V-3.4.1.a Library Installation Support

It is clear that each installation should have a well defined procedure to install and support the PSL. Rather than have each small contractor initiate his own procedure the recommendation is made to provide detailed guidelines to the small contractor for set up and installation of the PSL. The guidelines should contain specific examples which provide a clear understanding of exactly what is required.

The installation of the functional PSL elements by the small contractor on his in-house computer should be relatively simple. The source files would be supplied by the Government and the small contractor would merely have to compile the source on his computer after minor modifications to specifically stated parameters such as the default terminal DSRN. Support could be offered indirectly by the Government by retaining the producer of the source code on a consulting basis to provide direct aid to the small contractor who desires to install the functional PSL on his in-house system. Multiple versions could be retained by the Government as they are developed for each computer manufacturer's operating systems to be distributed to the small contractors for their various in-house systems. As the number of versions grow to cover all existing operating systems the consultant source-code writer would be gradually phased out.

Commercial time sharing houses are now offering sophisticated services to their subscribers. They are quick to respond to user needs and it can be expected that they will support PSL functional capabilities soon after the requirement is finalized. As can be seen from the examples presented in this section, many of the PSL functions are already available.

Any Government computer system that the small contractor accesses would be expected to already have the PSL capability installed since it would be a Government standard.

SPS V-3.4.1.b Library Maintenance Support

The small contractor is fully familiar with random access storage of on-line files with back-up on magnetic tape. It has been the procedure of this contractor to allocate complete volumes on the in-house computer to specific projects. The dynamic allocation of the volume is then left to the operating system which automatically reallocates the storage as the library is updated. When a project is completed, a final version is stored on master tapes to relieve the volume for a new project.

When accessing commercial time sharing systems, the procedure is quite similar. An entire user identification is assigned to a project. The operating system dynamically reallocates the storage when any files are removed or updated. When a project is completed backup tapes are generated.

The Government system offers dynamic allocation and reallocation of storage. When it is found that files are no longer needed they are removed, thus freeing storage. Tapes are prepared at the completion of each project.

The final magnetic tapes are, of course, not the only tapes created during the program development. At predefined periods of time the developing system is copied to magnetic tape. These backups, although not often recalled, are very inexpensive insurance against the loss of many hours of work. They provide a copy of the system in the event of catastrophic failure of the random access device.

SPS V-3.4.1.c Library Termination Support

Library termination support is provided by the storing of the PSL on magnetic tape and the removal of the PSL from the on-line storage.

SPS V-3.4.2 Full Requirements

SPS V-3.4.2.a.1 Hardware Configuration Definition

The in-house hardware configuration is well defined at the small contractor's site, while at commercial time sharing installations the hardware definition may be more fluid since only the portion utilized needs to be paid for. Similar procedures are used at Government installations. Installing a PSL at any of the sites would be relatively easy.

SPS V-3.4.2.a.2 Limited PSL Generation

A system generation facility that allows a programmer to generate a PSL that contains only the major functional capabilities required at his installation would only be applicable to the in-house system of the small contractor. It should be assumed that both the commercial time sharing system and the Government system would have full capabilities and the small contractor would utilize only the portions needed. By properly modularizing the PSL the small contractor would be able to install only what is required. It is recommended that the PSL supplied by the Government therefore be fully modularized to accommodate this facility.

SPS V-3.5 Data Security

SPS V-3.5.1 Basic

The ability to recover from inadvertent loss or destruction of data through the use of magnetic tape backups has been demon-

strated earlier on all three systems that the small contractor is most likely to access. Each system also has the capability to prevent inadvertent destruction of data as a result of an updating operation. The in-house system editor has the edit back feature that renames the input file by changing the file extension to BAK. The output file automatically takes on the name of the input file. Thus the original file remains unchanged. The commercial time sharing system has various options to retain an original version of a file while making updates. Each method requires that the name of the file being updated is changed. The first method would be to invoke the ALTER command to change the name of the file to be updated. The most convenient naming procedure is to append a digit to the file name indicating the version number. After the file has been edited and updated, the exit from edit can be made with the command FILE NEW FILE where NEW FILE is the new file name. If the original file already has a version number appended to it, the procedure would not require the ALTER command. An alternative approach would be to use the COMBINE command to rename the original file into a new file name. This would be the backup file. Then editing would occur on the original file and exit would be accomplished with the standard FILE command. This method has the advantage that the latest version of the file can always be found under the same name while previous versions are still available to the user.

The Government computer system has an on-line backup capability. By referring to the figures where the editor was first introduced to alter the contents of the PSL files the reader will note that the files that were catalogued under the same name as the original file each showed an incremented cycle count. This system allows five active cycles of a file which means the most recent version will be stored under the highest cycle number and up to four previous versions may be accessable to the programmer. Actually, five active cycles are too many because the storage costs become prohibitive for the amount of on-line storage required. The older versions are usually retained on magnetic tape.

None of the systems prevent the user from adding files of the same name to the system. The systems assume that if the user duplicates a name already in existence that he intends to replace the old file with the new.

SPS V-3.5.2 Full Requirements

SPS V-3.5.2.a Data Integrity

Data integrity between versions can be maintained on all three example systems available to the small contractor. In-

house integrity is provided by appropriately naming his files by version number. LINKing various program segments is performed by specifically naming the files to be included. By selecting the appropriate version to include he creates either the operational, development or maintenance version. For example, assume that a development version is being created of a system that is to include a new version of program B while utilizing the operational versions of program A and C. The files would be specified by their version numbers, assuming version 3 is the operational version and version 4 is the development version. Figure II.3.2-73 shows how this would be accomplished.

```
.R LINK
*B55VR4=B55AV3,B55BV4,B55CV3/F
*
```

Figure II.3.2-73. Data integrity between versions of a system on an in-house computer.

The commercial time sharing system can preserve the integrity between versions by assigning each version to a separate userid. The userid that contains the production version can be attached in read-only manner from the other userids. The development ID can then search the production ID for any needed files. An example is shown in Figure II.3.2-74. The development ID possesses a new version of program B and does not contain a version of programs A or C. A new development version load module is to be created utilizing the three programs. From the development ID the production ID is attached as a T disk. The load command is given. It automatically searches the development ID disk for the text files for programs A, B and C. Finding only B it proceeds to the T disk and finds programs A and C. Any system routines needed are then searched for on the system disk.

```
15.39.42 >ATTACH PRODUCT 191 AS 192 RO
PRODUCT ATTACHED AS 192,(RO)

15.39.53 >LOAD A B C

15.40.33 >GENMOD

15.42.15 >
```

Figure II.3.2-74. Example of preserving data integrity between development and production versions of a program on a commercial time sharing system.

When using the Government computer the user can assign distinct names to each version. When creating the load module for a new development version he need only attach the appropriate file for inclusion in the copy command before executing XEQ, Figure II.3.2-75.

```
COMMAND- ATTACH,A,APRODUCTION,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- ATTACH,B,BDEVELOP,ID=GAERTNER

 PF CYCLE NO.= 003
COMMAND- ATTACH,C,CPRODUCTION,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- COPYBR,A,S,99

  END OF FILE ENCOUNTERED AFTER COPY OF
       RECORD 3
COMMAND- COPYBR,B,S,99

  END OF FILE ENCOUNTERED AFTER COPY OF
       RECORD 2
COMMAND- COPYBR,C,S,99

  END OF FILE ENCOUNTERED AFTER COPY OF
       RECORD 10
COMMAND- XEQ,LOAD=S,NOGO
```

Figure II.3.2-75. Example of preserving data integrity between program versions on a typical Government computer.

SPS V-3.5.2.b Data Protection

SPS V-3.5.2.b.1 Restricted Access

Data protection in the form of restricted access to the data files of libraries within a project is available in varying degrees on all three systems most often used by the small contractor. The in-house system is rather basic. It requires that the volume containing the library be physically removed from the system and appropriately stored in a secure place. When using a commercial time sharing system, the restricted

access is visible from the time of system dial up. The user must know a minimum of three items before he can access a particular library. They are the computer on which they are stored, the userid and the password associated with the userid. The computer is usually known to anyone involved with the system. The userid would be known to all programmers who need to access it. The password which is not distributed is, as its name suggests, a key to the library. It consists of one to eight characters. Any character that can be typed at a terminal and recognized by the computer can be utilized in the password. There are additional safeguards that can be exploited, the first of which is a PROFILE exec. The PROFILE exec is a program that is executed each time the userid is logged in. The security aspect of the program is that it may ask for additional passwords. If these passwords are not entered properly, the userid can be logged off the system by the program. The second safeguard is a program called PROTECT exec. This program is executed each time the disk is attached by another userid. It also has the capability to ask for additional passwords. Furthermore, it can restrict the use of any or all computer commands, facilities, programs and data files. These restrictions can be based on the attaching userid, or the password entered. The mode of attach can be checked and certain users can be allowed to only update or read the disk. The PROTECT exec can also restrict access by the accounting information of the attaching userid. With the capabilities of PROFILE and PROTECT it can be seen that security of the library can be easily set up and maintained.

When using the Government computer system the programmer must again enter a userid and a password. Once on the system the names and associated userids of all files in the system can be ascertained by use of the AUDIT command. This does not give the user access to the files though. Each file may be assigned up to five distinct passwords which give varying degrees of access privilege. Data security can be maintained for individual libraries in this fashion.

SPS V-3.5.2.b.2 Output Access, Restricted Update

Free access to data units for output but restricted update privileges can be achieved on either the commercial time sharing system or the Government computer. In this mode of operation, the libraries or files, depending on the system, are given passwords for "updatable", "writable" and "read only". The read-only password is given to specified individuals who only output the files from random access disk to other media such as printer or magnetic tape.

SPS V-3.5.2.b.2.c Output and Update Access

Free access to data units for either output or update but not extending can be accomplished on either the commercial time sharing system or on the Government system. Again, a password is set up for update privileges and distributed to the appropriate individuals. With update privileges the user can read the files and make changes to the files by replacement of any data that exists in them. It is not possible for the user to extend the file though.

SPS V-3.5.2.b.3 Classified Title Printing

Printing classified titles such as confidential or secret is not presently available on any of the three example systems most often used by the small contractor. This capability is not desirable for commercial time sharing systems because they do not offer the security required by Government standards for these levels. The majority of user interaction is carried on through a remote terminal that accesses the computer via common telephone lines. When Government computers are employed, the same telephone access is available. However, on-site time can be allocated for various levels of security classification. The first step to achieve the minimum security level is to disable the remote access capability. Then the computer is cleaned to remove any latent programs before it is exercised in a dedicated mode. The in-house system of the small contractor often does not have remote access capability but strict security rules must be followed to achieve classified status. It is assumed the small contractor could request the Government computing system for classified work and that the PSL capabilities would be available to print the desired titles. The in-house system would require the implementation of the PSL capability for classified title printing. It is recommended that the ability to print classified titles be implemented as an element of the PSL distributed by the Government but that it be supplied only on an as required basis and after the particular contractor has met the other Computer Facility clearance requirements.

SPS V-3.6 Management Data Collection and Reporting

SPS V-3.6.1 Basic

There are no basic requirements.

SPS V-3.6.2 Full Requirements

SPS V-3.6.2.a Collection

SPS V-3.6.2.a.1 Counting input source code

Counting input source code is not directly available on either the Government or in-house system. The commercial time sharing system does this automatically in its disk directory. It is recommended that an element of PSL be developed to count the number of lines of source code input for a unit, to be distributed by the Government to small contractors. In order to make the process automatic it would most likely consist as a portion of a complete operating system. It is impractical for the Government to supply full operating systems to small contractors so it is recommended that the function be non automatic but invokable on an as required basis.

SPS V-3.6.2.a.2 Gathering and Storing Source Unit Date Data

Gathering and storing source unit start date, end date and other similar data, which is available to the PSL in a management-statistical data base, cannot be accomplished on the present systems under discussion. In its present form the in-house system only returns the start date for a source code file. The commercial time sharing system has the start date and the date of the last alteration which would be the end date of the source unit. The Government computer has available the starting and ending dates for source files. However, there is no method to collect this information on specific files and to store it in a meaningful manner in a table.

SPS V-3.6.2.a.3 Counting PSL Functions Used

Retaining a count of the number and type of functions invoked from the PSL is not readily available automatically. Items such as the number of compilations of source file could be counted manually but automatic retention requires more of a system monitoring package. One suggested method is to modify existing minicomputer operating systems to automatically record the terminal session on a disk file. The file could then be processed by an accounting program to develop statistics about the session. When accessing a large-scale time sharing system a terminal that has the capability to simultaneously record and communicate such as the CMC/2741 would be utilized. The recorded data could then be processed through an accounting program.

SPS V-3.6.2.a.4 Ability to Add Routines to the PSL

There is no lack of ability on the part of the small contractor to add routines to a PSL to satisfy unique requirements. The complexity of adding functions will be highly dependent upon how the PSL is initially configured. It is recommended that a small contractor be the prime contractor for the final design and implementation so that the needs of the majority of small contractors will be taken into account.

SPS V-3.6.2.b Updating

SPS V-3.6.2.b.1 Storage Allocation

Allocating management statistical data storage areas is done automatically on all three example systems when the files are created.

SPS V-3.6.2.b.2 Management Statistical Data Editing

Editing of management statistical data could be accomplished by the editors on the systems so far discussed. The programmer would not be notified of any format or data errors by this method until an actual report was produced. It is recommended that a Management Statistical Data Handling (MSDH) routine be developed that would allow input of data in English format and validation of the input information.

SPS V-3.6.2.b.3 Adding Management Statistical Data

Adding management statistical data supplied by the user could also be a function of the Management Statistical Data Handler recommended in the previous paragraph.

SPS V-3.6.2.b.4 Deleting Management Statistical Data

Deletion of user supplied information could be handled by the MSDH.

SPS V-3.6.2.b.5 Replacing Management Statistical Data

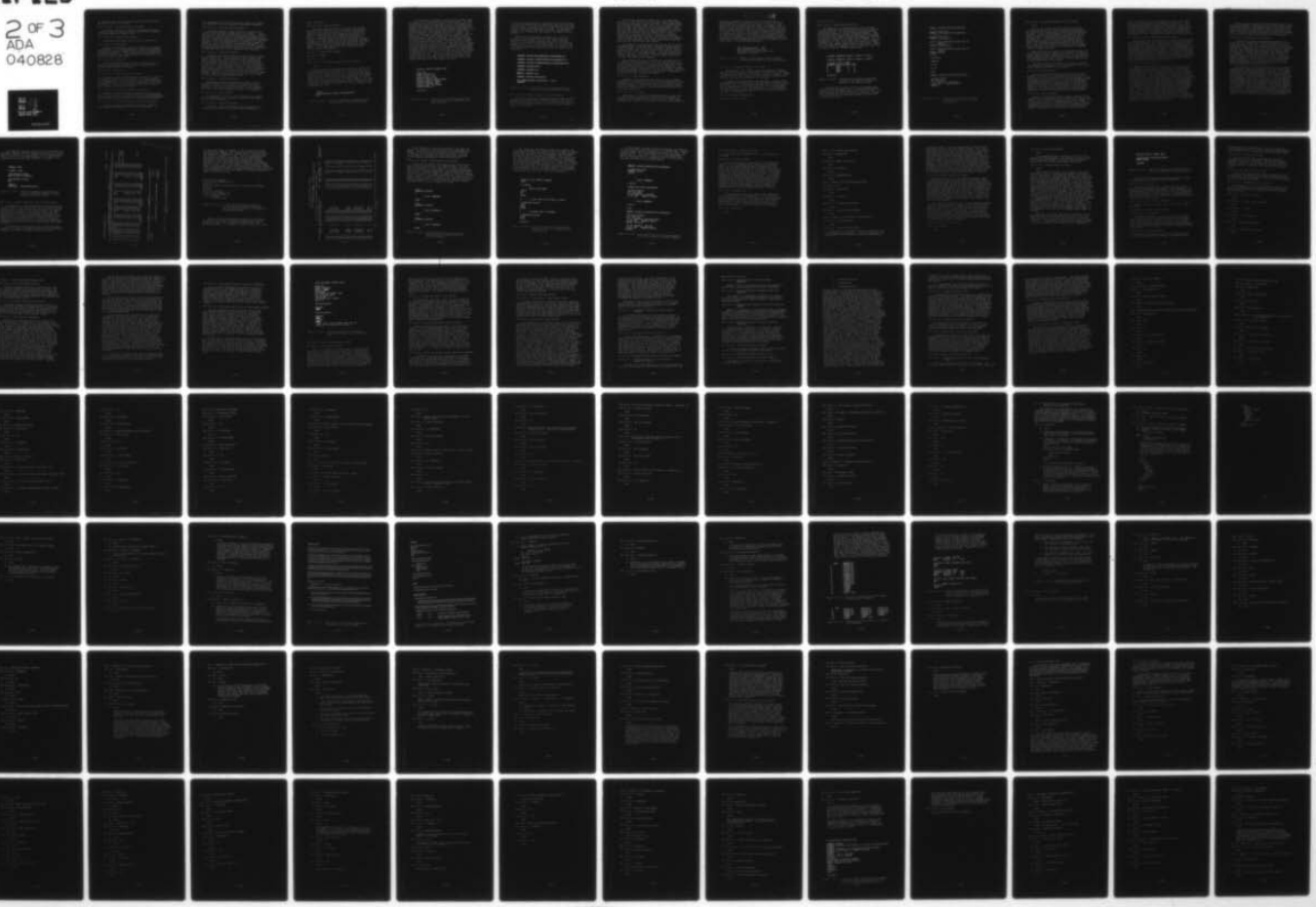
Replacing data in the management statistics data base should be a function of the MSDH.

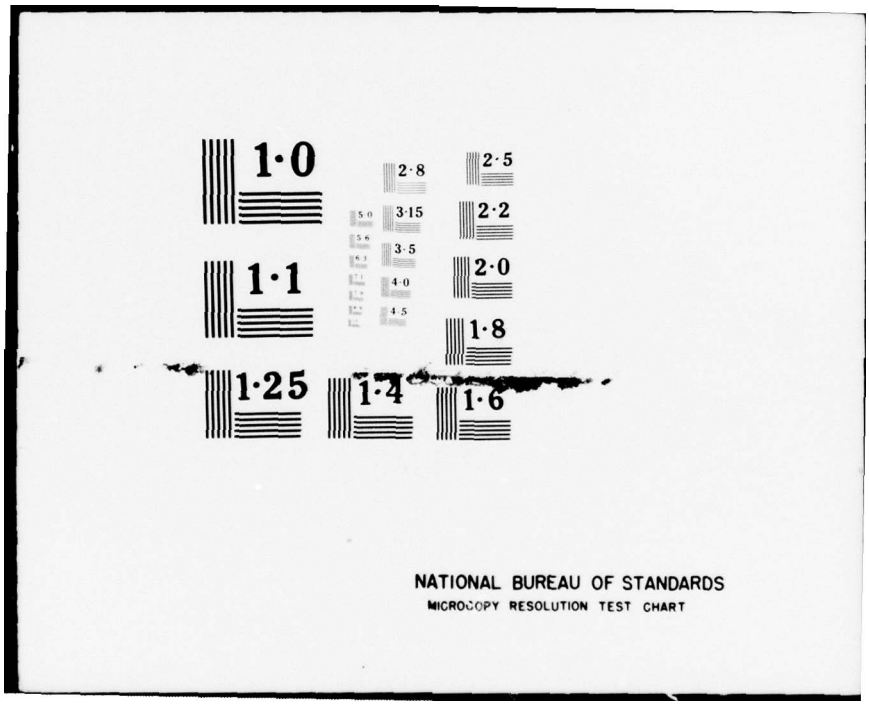
SPS V-3.6.2.b.6 Storing Computer Turnaround Time Data

Storing data relating to computer turnaround time is not as critical in the real-time environment which most small contractors use. Perhaps system response time could

AD-A040 828 GAERTNER (W W) RESEARCH INC STAMFORD CONN F/G 5/1
IMPACT OF STRUCTURED PROGRAMMING STANDARDS ON SMALL GOVERNMENT --ETC(U)
MAY 77 W W GAERTNER F30602-76-C-0390
UNCLASSIFIED RADC-TR-77-162-VOL-2 NL

2 OF 3
ADA
040828




1·0
2·8
2·5
3·15
2·2
3·5
1·1
2·0
4·0
1·8
1·25
1·4
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

be logged and the chart would then show minutes and seconds not hours or days as for batch.

SPS V-3.6.2.b.7 Terminating Data Collection

Providing a function of the PSL to terminate the collection of data for specific types of data such as computer utilization would be a manual task.

SPS V-3.6.2.b.8 Terminate the Project

Terminating the project would also be a manual task.

SPS V-3.6.2.c Accumulating

Summarizing management statistical data is not presently available. It is recommended that a program be developed to perform the task of summarizing the data and outputting it in meaningful reports. There should be options available to the user to specify what items mentioned in subsection 3.6.2.c will be included in the report.

SPS V-3.6.2.d Archiving

It has been demonstrated that any information that is stored on the example computers could be transferred to magnetic tape for offline storage.

SPS V-3.6.2.e Reporting

SPS V-3.6.2.e.1 Module Statistical Reports

Producing program module statistical reports for several hierarchical levels is not available on the three systems. The recommendation is made to include such a report generator as an element of the PSL provided by Government to the small contractors. It would use the accumulated data file as input to produce the required reports.

SPS V-3.6.2.e.2 Reports of Computer Utilization

Computer utilization reports containing information related to the analysis of turnaround time for each programmer and a summary for the programming staff would not be applicable to the small contractor working in a real-time environment.

SPS V-3.6.2.e.3 Program Maintenance Statistical Reports

Producing program maintenance statistical reports is not available automatically to the small contractor. It

is recommended that program maintenance statistical reports be a function of the PSL supplied to the small contractor.

SPS V-3.6.2.e.4 Program Structure Reports

Producing program structure reports is assumed to be similar to the automatic production of flow charts. There are some very good flow charting programs available. However, they do not presently conform to structured programming restraints. With the advent of structured programming inevitably various programs will be developed to document the structured programs. It is recommended that the Government support an effort to develop an Automatic Structured Programming Tree Generator (ASPT) that would be the standard of the industry. The program could then be made available to small contractors so that they could use ASPT to document their programs.

SPS V-3.6.2.e.5 Historical Reports

Producing historical reports from the archival projects data by means of an inquiry capability would require additional programming. The description of the search parameters supplied indicates a massive volume containing the historical data from many contracts. The small contractor is more likely to approach the storage of historical data on the basis of a single volume per project. This technique might suggest that a continual stream of magnetic tapes would have to be mounted for the inquiry system to complete the request. This would not necessarily be true since hard copy in human readable form would be available for each project to help select the needed magnetic tapes.

SPS V-3.6.2.e.6 Combination Reports

Combination reports could also be produced from the various statistical files maintained by additional programming. Although not presently available, the programming effort could be reduced considerably if the Government supplied a basic report generator, which is recommended.

SPS V-3.6.2.e.7 Cyclic Reports

The production of cyclic reports should be an element of the recommended report generator supplied by the Government to small contractors.

SPS V-3.6.2.e.8 Unique User Reports

Unique user report requirements could be added to the standard report generator supplied by the Government to

small contractors.

SPS V-3.6.2.e.9 Report Generator

The report generator that is to follow the requirements set out in Section 3.6.2 of Volume V would be most valuable to the small contractor if it were interactive. Supplying English inputs would generate any desired reports. To complete the report generation session a completion instruction such as FINISHED or END would be entered. The report generator would then produce a table of contents of the reports and if any significant variances between actual and plan data had been observed by the program, it would produce an additional directory to those reports. Also, the items found to be beyond specified limits could be highlighted.

SPS V-3.7 Documentation Support

SPS V-3.7.1 Basic

SPS V-3.7.2 Full

SPS V-3.7.2.a Printing Specified Files of the PSL

Clear

Printing specific sequences of sections of the PSL and merging selected portions of the PSL into one print file is available on all three systems. The in-house system would utilize PIP to produce the listings. PIP accepts the output device and any number of files to be outputted. The example shown in Figure II.3.2-76 gives the output command. LP is specified as the output device, i.e. line printer. The files output are a program design language file, a source file that implements the PDL and a textual file that is related to the source and PDL files.

```
.R PIP
*LP:=B55CMA.PDL,B55CMA.FOR,B55CMA.TEX
*^C
```

Figure II.3.2-76. Printing a specified sequence of files on the typical in-house minicomputer.

The commercial time sharing system offers greater capabilities to print selected files of varying types. It might be thought that each file directed to the line printer on a large system would be immediately placed in the output queue. However, the commercial time sharing system allows the programmer to selectively stack up any number of individual files before they are entered in the queue so that they are all printed together. The command to do this is CLOSIO. The device, and whether to hold the output or release it must be specified. An additional feature of the commercial time sharing system is the TITLE command. This allows the user to specify up to three lines of information, with a maximum of eight characters each, which will be printed two inches high. The title is printed twice so that one copy is visible no matter how the listing is folded. When the title command is issued between listings it is recommended to include the optional EJECT parameter which performs a top of form before printing starts. Figure II.3.2-77 contains a stored sequence of commands set up to print a PDL file, a source file and a related textual file. Each is preceded by a title that includes the file name, the type of file and the date. The CLOSIO command is included in the file so that the printed output will consist of the three files only.

```
13.12.46 >PRINTF DOCUMNT EXEC

&TYPE OFF
CLOSIO PRINTER OFF
TITLE &1 PDL &DATE
OFFLINE PRINT &1 PDL
TITLE &1 SOURCE &DATE (EJECT)
OFFLINE PRINT &1 SOURCE
TITLE &1 TEXTUAL &DATE
OFFLINE PRINT &1 TEXTUAL
CLOSIO PRINTER ON

13.14.22 >
```

Figure II.3.2-77. Printing a user-specified sequence of files from a PSL on a commercial time sharing system.

To use the stored sequence of procedures the user must type its name followed by the file name associated with the files to be printed. The symbol &1 will be automatically replaced in each line with the specified file name. If the above sequence is not desired then the appropriate combination of commands may be entered at the terminal to produce similar results.

When using the Government computer the files to be printed must be combined into one file before they are directed to the printer to ensure they remain together. Figure II.3.2-78 demonstrates how three files are accessed, combined and printed. The COPYSBF command is used for combining the files to shift the actual data one column to the right because the first column is used as a printer control character.

```
COMMAND- ATTACH,PDL,SAMPLEPDL,ID=GAERTNER,CY=1

COMMAND- ATTACH,S,SAMPLESOURCE,ID=GAERTNER,CY=3

COMMAND- ATTACH,TX,SAMPLETEXTUAL,ID=GAERTNER,CY=2

COMMAND- COPYSBF,PDL,0

COMMAND- COPYSBF,S,0

OMMAND- COPYSBF,TX,0

COMMAND- BATCH,0,PRINT,GAER

FILE NAME-IGAERB3 ,DISP-PRINT  ,ID-**
COMMAND-
```

Figure II.3.2-78. Printing a user specified sequence of files on a typical Government computer.

SPS V-3.7.2.b Automatic Output Listing Formatting

Automatic formatting of output listings by user directed commands has been partially implemented by this small contractor on his in-house system. The commercial time sharing system and Government computer offer an alternative method that is less automatic.

The in-house program is called LIST. The input parameters are: The file to list, the output device, the number of lines per page and a line of notation. The program prints the file with a double header line. The first line includes the file name, the date, the time and a page number. The second header line contains up to 60 characters of annotation that was entered by the user. Spacing between lines of output cannot be changed since the content of the file is printed as is, and only one file as opposed to an entire library can be printed at a time. The above two modifications could be made relatively easily to attain full implementation of the paragraph. With slight modification this program could run on the other two systems.

The commercial time sharing system already offers partial implementation by using the PRINT or PRINTCC feature of the OFFLINE command. The PRINT option produces a header which includes the file name, the userid, which would be the library name, the date, time and page number. Each page contains 55 lines of the file. The PRINTCC feature assumes that the file has been formatted by the user. It uses column one of the file as a carriage-control character.

When printing a file on the Government computer the file is assumed to be preformatted. If it is a raw source file, it must first be copied using the COPYSBF command since column one is assumed to contain a carriage control character. Unfortunately, the raw source file will not be broken into pages but will be printed continuously from the first page. The printing can be enhanced if the file to be printed is generated by a compiler that has formatted the output.

SPS V-3.7.2.c Page Numbering

Automatic page numbering beginning with a user supplied page number is not available on any of the systems but if it is desired to start with page number one the LIST program may be utilized in-house, or the OFFLINE PRINT command would suffice on the time sharing system. The Government computer does not offer page numbering.

SPS V-3.7.2.d Title Page

Printing of a title page containing document title, document date, author's name(s) and organization and address is possible on the commercial time sharing system. The system automatically includes at the front of each listing an organiza-

BEST AVAILABLE COPY

tion name and address which the user has prespecified in a special address file as shown in Figure II.3.2-79. By using the OFFLINE PRINT command the file title is included at the heading of each page along with the date. The author's name(s) may be included by use of the TITLE command. If desirable, the TITLE command may also be used to put a document title and date in front of the listing. The in-house system has no capability to generate the specified information on a title page nor does the Government computer. This could be implemented by creating such a file and printing it before the document was printed. Extensions to the in-house program LIST could automate this procedure which is the recommended approach.

```
MAIL FIRST CLASS TO     WMS
W.W. GAERTNER RESEARCH INC.
1492 HIGH RIDGE RD. 2ND FLOOR, RM 1
STAMFORD, CT. 06903
```

Figure II.3.2-79. Example of organization name and address printed by the commercial time sharing system.

SPS V-3.7.2.e Magnetic Tape in Print Image

The commercial time sharing system can generate a magnetic tape in print image form. The program called UTILITY is utilized. The tape would be created by the disk to tape, DT, feature. After creation in this manner the TL or tape-to-listing command can be exploited to print the information on the tape.

In-house, this capability is not available. However, with some extension, the program LIST could generate print-image files on magnetic tape. The Government computer does not offer generation of magnetic tape in print-image form. It is recommended that the Government support the production of a listing program for small contractors that implements the features set forth in SPS Volume V, Subsection 3.7.2.

SPS V-3.8 General Requirements

SPS V-3.8.1 Basic - none

SPS V-3.8.2 Full

SPS V-3.8.2.a Concatenated Data Files

The capability to concatenate data files into one file is available on all three systems by various forms of a COMBINE function. On the in-house system PIP would be the utility program for accomplishing the COMBINE. A new file would be created that was a combination of any number of existing files. When accessing the commercial time sharing system, a command is available called COMBINE. It can concatenate into one file as many files as can be listed on a single line. If all files cannot be listed on a single line, then the COMBINE command can be issued again. Figure II.3.2-80 shows the creation of file D from the three files A, B and C using the COMBINE command twice.

```
15.32.02 >COMBINE D DATA P A DATA P B DATA P

15.32.48 >COMBINE D DATA P D DATA P C DATA P

15.34.05 >LISTF * DATA
FILENAME FILETYPE MODE ITEMS
A        DATA       P     4
B        DATA       P     4
C        DATA       P     4
D        DATA       P    12

15.35.24 >
```

Figure II.3.2-80. Concatenating data files by multiple issuance of the COMBINE command on a commercial time sharing system.

The Government computer could also concatenate data files into one contiguous file. The COPY command would create the new file which consisted of the old files. The new file would then be the input to the program. Figure II.3.2-81 contains the commands necessary to create File C from files A and B.

```
COMMAND- ATTACH,A,ADATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- ATTACH,B,BDATA,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- COPYCR,A,S,9

 END OF FILE ENCOUNTERED AFTER COPY OF
        RECORD 1
COMMAND- COPYCR,B,S,9

 END OF FILE ENCOUNTERED AFTER COPY OF
        RECORD 1
COMMAND- EDITOR
..E,S,S

..L,/EOR/,A

 630=*EOR

..L,630

..S,C,N

..BYE

COMMAND- CATALOG,C,CDATA,ID=GAERTNER

 INITIAL CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=CDATA
 CT CY= 001     0000312 WORDS.:
COMMAND-
```

Figure II.3.2-81. Concatenating data files into one contiguous data file on a Government computer.

SPS V-3.8.2.b Subroutine Support for User Program

Clear

The PSL can provide subroutine support for any of the three systems as long as its source code is supplied. All three of the systems support libraries loaded with user programs. On the in-house system, the source of the PSL would be compiled and the object files would be put into a library file. A programmer requiring one or more of the functions of the PSL in one of his programs would merely have to specify the library name to the LINK program. The LINK program would automatically search the PSL library file for unsatisfied externals to be included in the load image file much like it searches the FORTRAN, COBOL or systems libraries.

When accessing the commercial time sharing system, the source code for the PSL might not be necessary if the particular system already supported the PSL. In this instance the programmer would possibly have to attach a special disk that contained the library in proper form and then specify it when generating his load module.

The small contractor accessing a Government-installation computer would be able to use the library since it most likely would be supported in the system.

SPS V-3.9 On-Line Terminals

This small contractor has not reviewed this series of documents in order to join the debate, if there is still one going on, over the advantages of on-line terminal processing versus batch processing. It should be pointed out that the examples presented so far have only referenced batch processing when the time sharing system did not have the ability to accomplish a function directly. All of the examples are a result of on-line terminal interaction with a computer. The majority of requirements so far presented, both basic and full, have been met by the three systems.

SPS V-3.9.a Terminal Support

Support of terminals which are either connected to the computer via communication links or directly via cables and control units is already available on all three systems. The in-house system at W. W. Gaertner Research, Inc. supports two hardwired terminals. This basic configuration could be expanded by utilizing standard options available from the computer manufacturer to allow access by more terminals

and/or to permit access via communication links. The commercial time sharing system offers greater capabilities. The system may be accessed via local telephone calls from major cities throughout the United States, Canada and Europe. Over 700 users may access the system simultaneously. Remote batch is also available on the system. When using the Government computer, access is via hardwired terminal or dialable telecommunication links. Remote batch processing is also available.

SPS V-3.9.b CRT and Typewriter Terminal Support

Both display (CRT) and typewriter terminals are supported on all three systems. The in-house system can serve CRT terminals operating at various speeds from 110 to 9600 baud and most other ASCII terminals. Other types of terminals can also be connected to the in-house computer, such as an IBM 2741. The commercial time sharing system offers connection capabilities for both CRT and typewriter terminals. The speed at which the terminal operates is user selectable from 10 CPS up to 120 CPS. When using the Government computer the line speed is determined by dialing the appropriate access number for either 10 or 30 CPS. The actual type of terminal may be either CRT or typewriter as long as it communicates in ASCII.

SPS V-3.9.c Remote Job Entry

Remote job entry to initiate batch PSL jobs from an online terminal is available in various forms on the systems. The in-house system allows the programmer to set up a job stream and execute it in a batch environment. This capability is exploited to run jobs unattended for long periods of time such as overnight. When accessing the commercial time sharing system, two forms of batch processing are available. The first is to set up a job stream that would most often be interactively controlled from a time sharing terminal. The programmer directs the computer to batch this job and to either execute it immediately or, more often, to execute the job overnight. The overnight option specifies that execution will not begin until system utilization drops below a certain level after 6 P.M. The advantage is that lower rates are charged at this time. The user receives the results of the job or jobs the next morning. The second alternative is to connect a remote batch terminal to the computer via a communication link. The small contractor does not often use this mode because of the additional hardware involved. It is more cost effective to use direct terminal access.

Previous examples showed the batch capabilities of the Government computer. Some function must be performed in batch such as those that access magnetic tape or the line printer. If the user desires he may set a priority on the batch job specifying that it be processed in the late evening at lower rate. It is also possible to connect a remote batch terminal to the system to input cards and receive printer output.

SPS V-3.9.d Interterminal Communication

A terminal to terminal communication capability to allow users to exchange data on a PSL system can be interpreted two ways. The first is the capability for two programmers to communicate via messages to each other's terminal. The second is that they are able to pass files to each other. Both will be discussed concurrently on the systems. When utilizing the in-house system interterminal communication via messages is accomplished verbally. The proximity of the terminals is such that this is possible. There is no specific capability on the system for printed communication between terminals. When files are to be accessed by more than one programmer, it is not necessary to perform an actual transfer since only one PSL is mounted at a time which is accessable by both programmers.

The commercial time sharing system allows both modes of communication. When a user wishes to talk to another user the MSG, meaning message, command is engaged. This command identifies the proposed recipient and the textual message. To send data files the sender directs his output device to the recipient via the XFER command. Then output is directed to the device. The example in Figure II.3.2-82 shows how userid GAERTNER receives a message from userid GRI requesting file SAMPLE DATA. Userid GAERTNER responds to GRI with a verification via the MSG command. Then the XFER command is invoked to direct userid GAERTNER's card punch, device D, to userid GRI. Other valid devices are the line printer or the console. Next, the OFFLINE PUNCH command is issued to send the file. Then the XFER OFF command is entered to disconnect the file communication link previously established. Figure II.3.2-83 contains userid GRI's terminal session.

```
08.32.41 >
FROM GRI: PLS SEND SAMPLE DATA

08.33.06 >MSG GRI OK, WILL XFR

08.33.48 >XFER D TO GRI

08.34.03 >OFFLINE PUNCH SAMPLE DATA

08.34.29 >XFER D OFF

08.34.43 >MSG GRI DID U GET IT?

08.35.12 >
FROM GRI: YUP
```

Figure II.3.2-82. Terminal-to-terminal communication capability of a commercial time-sharing system showing both message and data file transfer.

```
08.32.23 >MSG GAERTNER PLS SEND SAMPLE DATA

08.33.48 >
FROM GAERTNER: OK, WILL XFER

CARDS TO BE READ

>OFFLINE READ *
OFFLINE READ SAMPLE DATA

08.34.33 >
FROM GAERTNER: DID U GET IT?

08.34.54 >MSG GAERTNER YUP
```

Figure II.3.2-83. Terminal-to-terminal communciation capability of a commercial time-sharing system showing both message and data file receipt.

The Government computer allows users to communicate via messages. However, the file transfer function is not necessary since the entire file structure of the system is available to all users. The SEND command is for interterminal communication as can be seen in Figure II.3.2-84.

```
COMMAND- SEND

TO WHOM - MJEK

TYPE MESSAGE OR END-
WHAT IS PPN OF COMPUTE A

TYPE MESSAGE OR END-
END

COMMAND-
FROM MJEK - PPN=B55CMPUTALPHA
```

Figure II.3.2-84. Terminal-to-terminal communication on a typical Government computer showing message sending and receipt.

SPS V-3.9.e Terminal Statistics Collection and Output

A capability to collect and output statistics related to the use of each on-line terminal is not available on the in-house system. Some of this information such as terminal time utilized is presently collected by hand. The types and number of requests is not presently collected. The terminal usage time might be collected by requiring a programmer to run two simple programs LOGON and LOGOFF which would record his utilization on a file that could be accessed by an accounting program. Counting and identifying programmer requests would be carried out by the system monitor program.

The commercial time sharing system already does basic accounting of system utilization by each user for accounting purposes. An example is shown in Figure II.3.2-85.

GAERTNER RESEARCH INC 11A140 GAERTNER

SC	DAY	LOGIN	CONNECT HRS	CONNECT AMT	COMPUTE VPUS	COMPUTE AMT	OTHER UNITS	OTHER AMT	TOTAL	ACCOUNT INFO
39	14	14:41	0.67	6.70	116.90	23.38	1000	1.30	31.38	4037
40	14	15:41	0.99	9.90	125.40	25.08	3800	4.94	39.92	4037
41	14	16:01	0.00	0.00	0.00	0.00	0	0.50	.50	14160128---PRINT
42	14	16:30	0.06	0.60	0.00	0.00	0	0.00	.60	TAPE 0282
43	14	16:39	0.00	0.00	0.00	0.00	0	0.50	.50	14163953---PRINT
44	15	9:03	0.12	1.20	0.10	0.02	100	0.13	1.35	4037
45	15	9:10	0.61	6.10	135.20	27.04	6400	8.32	41.46	4037
46	15	10:56	0.49	4.90	160.10	32.02	7000	9.10	46.02	4037
47	15	13:06	0.37	3.70	89.10	17.82	3100	4.03	25.55	4037
48	15	13:33	0.03	0.30	0.40	0.08	100	0.13	.51	4037
49	15	13:56	0.64	6.40	10.50	2.10	200	0.26	8.76	4037
50	15	14:45	0.60	6.00	134.00	26.80	4500	5.85	38.65	4037
51	15	14:53	0.14	1.40	0.00	0.00	0	0.00	1.40	TAPE 0282
52	15	16:02	0.41	4.10	100.40	20.08	3900	5.07	29.25	4037
53	15	16:55	0.30	3.00	110.10	22.02	4000	5.20	30.22	4037
				$167.74		$340.56		$69.04	$577.34	

DATE	DAYS	CYLINDERS	AMOUNT
01/01/75	15	4	40.89
	4 RENTED TAPE(S) AT $5.00 PER MONTH		9.75
			50.64

Figure II.3.2-85. Time-share cost allocation system used by W.W. Gaertner Research, Inc. for IBM 370/168.

BEST AVAILABLE COPY

The type and number of requests is not presently collected by the system. However, it is possible to do so by issuing the command SET CONSPOOL ON in which case all terminal input and output is recorded on a file. This file should then be directed to the user's card reader via the XFER command. The command to terminate collection of terminal data is VP CLOSE 9. Then the command SET CONSPOOL OFF is entered. Now the user has in his card reader a record of his terminal session in one or more files, each file with maximum of 12000 bytes. By processing these files through an accounting routine to be developed, the number and type of requests can be summarized into an accounting file. The sequence of commands necessary is shown in Figure II.3.2-86.

```
XX.XX.XX SET CONSPOOL ON
XX.XX.XX XFER CONSPOOL TO ME
XX.XX.XX
XX.XX.XX (TERMINAL SESSION USING ANY VALID SYSTEM COMMAND)
XX.XX.XX
XX.XX.XX
XX.XX.XX VP CLOSE 9
XX.XX.XX SET CONSPOOL OFF
XX.XX.XX XFER CONSPOOL OFF
XX.XX.XX COMACNT
       : (PROGRAM RUNS)
```

Figure II.3.2-86. Collecting and processing statistics for the type and number of requests during a terminal session on a commercial time sharing system.

When accessing the Government computer the accounting of terminal utilization is performed automatically by the operating system. The information is used to bill the accounting centers. An example is shown in Figure II.3.2-87.

6000 COST REPORT-PICATINNY ARSENAL

REPORT COVERING MONTH OF JAN 1976
WITH A PRIME/LOW HOURLY RATE OF $ 367.20/$ 244.80

BILL TO FT. MONMOUTH - COST CENTER MGR - GAERTNER RESEARCH INC.

MACHINE	PROGRAM NUMBER	TYPE TYPE	JOB NAME OR PF ID	REQUESTERS (OR PF) NAME / COMMENTS	DATE	CHARGE CODE	CDC 65/6600 TIME (MINS)		COST	TOTAL FOR EACH CHARGE CODE
6600			T8T8IT8	GGGG	1/ 5/76	333	2.46		$ 15.06	
6600			T8T8IT8	GGGG	1/ 5/76	333	1.37		$ 8.38	
6600			T8T8IT8	GGGG	1/ 5/76	333	1.34		$ 8.20	
6600	27661	P	GAERO1T	SCHREYER	1/ 6/76	333	.20		$ 1.22	
6600	27661	P	GAERO12	SCHREYER	1/ 6/76	333	.24		$ 1.47	
6600	27661	P	GAERO17	SCHREYER	1/ 6/76	333	.24		$ 1.47	
6600	27661	P	GAERO2A	SCHREYER	1/ 6/76	333	.20		$ 1.22	
6600	27661	P	GAERO59	SCHREYER.	1/ 6/76	333	.69		$ 4.22	
6600	27661	P	GAERO9Y	SCHREYER	1/ 6/76	333	.48		$ 2.94	
6600	27661	P	GAERO9U	SCHREYER	1/ 6/76	333	.49		$ 3.00	
6600			T8T8IT8	GGGG	1/ 6/76	333	3.83		$ 23.44	
6600			T8T8IT8	GGGG	1/ 6/76	333	3.33		$ 20.38	
6600			T8T8IT8	GGGG	1/ 6/76	333	1.60		$ 9.79	
6600			T8T8IT8	GGGG	1/ 6/76	333	.14		$ .86	
6600			T8T8IT8	GGGG	1/ 6/76	333	.05		$ .31	
6600	27661	P	GAERO1Z	SCHREYER.	1/ 9/76	333	.24		$ 1.47	
6600	27661	P	GAERO2B	SCHREYER.	1/ 9/76	333	1.63		$ 9.98	
6600	27661	P	GAERO2K	SCHREYER.	1/ 9/76	333	1.75		$ 10.71	
6600	27661	P	GAERO2T	SCHREYER.	1/ 9/76	333	.01		$ .06	
6600			T8T8IT8	GGGG	1/ 9/76	333	1.68		$ 10.28	
6600			T8T8IT8	GGGG	1/ 9/76	333	.05		$ .31	
6600	27661	P	GAERO1H	SCHREYER.	1/14/76	333	.25		$ 1.53	
6600	27661	P	GAERO1U	SCHREYER.	1/14/76	333	1.61		$ 9.85	
6600	27661	P	GAERO17	SCHREYER.	1/14/76	333	.02		$ .12	
6600	27661	P	GAERO2H	SCHREYER.	1/14/76	333	1.75		$ 10.71	
6600	27661	P	GAERO3E	SCHREYER.	1/14/76	333	.01		$ .06	
6600	27661	P	GAERO9E	SCHREYER.	1/14/76	333	.25	**	$ 1.02	
6600	27661	P	GAERO9G	SCHREYER.	1/14/76	333	.01	**	$ .04	
6600			T8T8IT8	GGGG	1/14/76	333	.69		$ 4.22	
6600			T8T8IT8	GGGG	1/14/76	333	.70		$ 4.28	
6600			T8T8IT8	GGGG	1/14/76	333	.33		$ 2.02	
6600	27661	P	GAERO1W	SCHREYER.	1/15/76	333	1.61		$ 9.85	
6600	27661	P	GAERO1O	SCHREYER.	1/15/76	333	.01		$ .06	
6600			T8T8IT8	GGGG	1/15/76	333	1.53		$ 9.36	
6600	27661	P	GAERQAP	SCHREYER.	1/26/76	333	.06		$ .37	

Figure II.3.2-87. Collecting and reporting terminal time on a Government computer.

The collection of statistics related to the types and numbers of requests is not performed on the Government computer. There is no method to record the terminal session as done on the commercial time sharing system. It is believed that such a capability would have to be supported by the computer manufacturer to be cost effective.

SPS V-3.9.f System Failure Recovery Facilities

A facility to recover from system failure with minimum loss of data is available on all three systems. The time when the programmer is most vulnerable to system failure is when he is in the edit environment. All operations performed while in edit are not permanently recorded unless the editor is exited or a special command is issued to record any changes made on the disk. This small contractor has a policy of saving editor changes at regular intervals during an edit session in order to avoid total loss of the session. When utilizing the in-house system, this is accomplished by regularly exiting from the editor as seen in Figure II.3.2-88.

```
.R E
*EBSAMPLE.DAT$R$$
*
        .
        . (EDIT COMMANDS)
        .
*EX$$

.R E
*EBSAMPLE.DAT$R$$
*
        .
        . (EDIT COMMANDS)
        .
*EX$$

.R E
*EBSAMPLE.DAT$R$$
*
        .
        . (EDIT COMMANDS)
        .
*EX$$
```

Figure II.3.2-88. Ensuring minimum loss of data while performing editor commands on an in-house minicomputer.

The commercial time sharing system has a feature that allows the programmer to retain all work performed in the edit environment by issuing the SAVE command. The SAVE command writes the edit buffer onto the permanent disk file and returns the user to the edit environment in the input mode. The FILE command could be used but the user is returned to the system level and would have to reissue the edit command to continue making changes to the file. Figure II.3.2-89 contains an edit session that replaces a stub with actual code. First, the stub code is deleted then the new code is input from two magnetic cards. The SAVE command is issued after the first card of 50 lines is input and the FILE command is issued after the second and final card is entered.

```
09.04.21 >EDIT SAMPLE FORTRAN
EDIT:

>L /WRITE/

        WRITE(IDIAG,1000)
>DE 99
EOF
>I
INPUT:
>       .
>       . (FIRST CARD OF 50 LINES IS INPUT)
>       .
> (CARRIAGE RETURN)
EDIT:
>SAVE
INPUT:
>       .
>       . (SECOND CARD IS ENTERED)
>       .
> (CARRIAGE RETURN)
EDIT:
>FILE

09.09.36 >
```

Figure II.3.2-89. Ensuring minimum loss of data while inputting a file on a commercial time sharing system.

The Government computer allows the programmer to interrupt his edit session at any time and save the changes made. However, the save command does not automatically retain the file if the system crashes. The intermediate file must be catalogued to ensure that it is not lost. Figure II.3.2-90 displays how the user would perform an intermediate save of a file and remove the file when it is no longer needed.

```
COMMAND- ATTACH,X,SAMPLEFORT,ID=GAERTNER

 PF CYCLE NO.= 001
COMMAND- EDITOR
..E,X,S

..        .
          .
          . (EDIT COMMANDS)
          .
..S,XT,N

..CATALOG,XT,TEMP,ID=GAERTNER

 INITIAL CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=TEMP
 CT CY= 001     0000312 WORDS.:
..        .
          .
          . (EDIT COMMANDS)
          .
..S,XF,N

..BYE

COMMAND- CATALOG,XF,SAMPLEFORT,ID=GAERTNER

 NEW CYCLE CATALOG
 RP = 090 DAYS
 CT ID= GAERTNER  PFN=SAMPLEFORT
 CT CY= 002     0000312 WORDS.:
COMMAND- PURGE,XT

 PR ID= GAERTNER  PFN=TEMP
 PR CY= 001     0000312 WORDS.:
COMMAND-
```

Figure II.3.2-90. Ensuring minimum loss of data during edit session on a typical Government computer.

SPS V-3.9.g and h. Security of Files

This subject was covered when Vol. V, 3.5.2.b.3 was discussed.

SPS V-3.9.i On-Line PSL HELP

To provide a PSL HELP file on-line is an attractive feature. It is available on the commercial time sharing system for various packaged programs accessable by the user. The Government computer also provides various HELP utilities for specific aspects of the system. The in-house system does not presently have any on-line HELP utilities. However, an equally beneficial aid to the user is provided in the form of abbreviated reference cards. It is recommended that reference cards be developed for the PSL along the same lines as the manufacturer's and commercial time sharing reference cards. Their application would not be limited to the on-line environment but would also be applicable to a batch environment. Typical examples of reference cards are IBM SYSTEM/360 Reference Data form number X20-1703 or PDP-11 programming cards from DEC.

SPS V-3.9.j Paging on CRT

The paging capabilities mentioned in the subsection 3.9.j are available in varying degrees. The in-house system could support an editor that had a full paging function but at present it does not. The commercial time sharing system supports limited paging to the CRT terminal via the CONSPOOL command. The Government computer supports many of the paging functions by a specific PAGE command.

SPS V-4.0 HIPO charts

Clear.

SPS VI PSL Program Specification

SPS VI-1.1 Introduction

Clear

SPS VI-1.2 Report Organization

Clear

SPS VI-1.3 Conclusions

Clear

SPS VI-1.4 Recommendations

Clear

SPS VI Section 2 Description/Requirements

SPS VI-2.1 PSL

SPS VI-2.1.1 Definition

Clear

SPS VI-2.1.2 Purpose

Clear

SPS VI-2.1.3 General Description

Clear

SPS VI-2.2 Functional Requirements

Clear

SPS VI Section 3 Environmental Considerations

SPS VI-3.1 Design Approach

Clear

SPS VI-3.2 Equipment Requirements

Subsection 3.2 of volume VI refers to Table 3-1, PSL Minimum Equipment Requirement. The small contractor has these capabilities available on either the commercial or

Government systems. In-house minicomputer systems could be configured to provide the stipulated hardware, but in many cases the facilities might be somewhat smaller. The 80 kilobyte requirement which does not include the operating system could be met by utilizing overlays to contain the PSL. A card reader as a minimum requirement should be excluded for the small contractor. The in-house system used by this small contractor does not have any card related devices, either reader or punch. We have experienced no drawbacks from their omission. The magnetic tape unit has proved to be more than adequate for both offline storage and program/data transfer to other computer systems. For instance, the transfer of the equivalent of 52 kilocards was accomplished at a cost of under $5 through the Postal service by using magnetic tape. The cost of actually using cards would have been much greater.

SPS VI-3.3 Software Support Environment

Operating the PSL as a self-contained software system is not the optimum method for the small contractor. When Volume V was discussed it was pointed out with examples that many of the functions of the PSL could already be accomplished. These functions are a part of existing operating systems that are assumed to be optimized. It would not be economical to duplicate the supported functions in a PSL. The capabilities not presently supported are the automatic collection of the system activities invoked such as the number of compiles or the number and type of edit functions. Also, the small contractor is more likely to employ milestones to measure the level of project completion as opposed to the number of compiles or edit requests.

The interface between PSL and "several external support programs" that are listed in Table 3-2 of Volume VI has not been fully developed. This probably results from the fact that the authors of the volume realize that the interface will be unique to each operating system. No matter what the size of a contractor, interfacing a packaged program into an operating system is not a trivial or one-time task. Operating systems are dynamic and get updated and improved. It is possible that such alterations to the operating system will have adverse affects on the PSL.

SPS VI Section 4 PSL Structure

SPS VI-4.1 General

Clear

SPS VI-4.2 Library Organization

Clear

It is recommended that a glossary of terms for the entire series be developed. For instance a "unit of data" has finally been defined in Volume VI, while Volume V applied this term extensively without definition.

SPS VI-4.2.1 Internal Data File Names

Clear

Internal data file names are available to the small contractor. On the in-house system they are referred to as file extensions. The commercial system calls them file types. The name is included in the 40 character identifier of the file on the Government computer. The first two systems mentioned have default file types for various files. It is recommended that the identifiers covered in paragraph 4.2.1 of Volume VI be called file types as opposed to file names because they specify the type of information contained in the file not the purpose of the file. The default file types that are provided on the two systems are in certain cases in conflict with the standard. For example, a standard file type of TEXT is for textual information while the commercial time sharing system utilizes the file type of TEXT to identify an object file by the terminology of the standard. It is recommended that the type TEXT be extended to TEXTUAL or changed to SCRIPT. The in-house system uses OBJ for object files so that there is no conflict. SOURCE is proposed to identify a program file. The disadvantage of this is that it does not identify the language of the code. It would be more meaningful to allow SOURCE to be changed to the language of the code such as FORTRAN, COBOL, JOVIAL or ASSEMBLY.

The LINK file assumes that control statements can be executed by the system program from a stored file. The small contractor can utilize the commercial time sharing system to do this by an EXEC file as can be seen in Figure II.3.2-91. In the example, three files are loaded and then a module is generated. The module is non-relocatable.

```
10.03.38 >PRINTF GENB55 EXEC

LOAD B55PROGA B55PROGB B55PROGC
GENMOD PROGA

10.04.05 >
```

Figure II.3.2-91. Sample of stored control deck to link 3 files on a commercial time sharing system.

On either the in-house or Government computer a batch run would execute the stored control statements.

SPS VI-4.2.1.g JOB File

The JOB file assumes that a stored sequence of commands can be executed by the computer. The commercial time sharing system as shown in the above Figure can do this, except that the type of the file is EXEC. On the in-house system, the job control is more likely to be contained within the main program while on the Government computer and a batch stream is set up and stored.

SPS VI-4.2.1.h and i TEST and MGMT Files

There is no conflict seen in the use of TEST or MGMT file types.

SPS VI-4.2.2 Internal Data Structure

SPS VI-4.2.2.a PSL Catalog

The small contractor involved in multiple PSLs would most likely store his PSL catalog offline. Each library would be contained as an entity. On the in-house system a specific volume would contain the library for each project. When accessing either the commercial time sharing system or the Government computer, a unique userid would handle each project.

SPS VI-4.2.2.b Project Catalog

The need of a project catalog implies that multiple libraries are accessed by a project, and therefore a re-

quirement exists to separate them. The small contractor could implement multiple library files by utilizing separate userids that could be linkable on the time sharing system.

SPS VI-4.2.2.c Library File Index

The library file index is available to the small contractor as a device directory. The device or userid directory contains a list of the units of data, presently referred to as files, on the volume or the userid for the in-house system or commercial time sharing system respectively.

SPS VI-4.2.2.d Name Table File Index

The optional name table file index to control access to a project is provided for at the small contractor's site by restricting the disbursement of the userids and passwords of the various library logins.

SPS VI-4.2.2.e Management Data File Index

The management data file index, which is optional, may not be utilized by the small contractor who employs a milestone-achieved technique to ascertain the level of completion of a project.

SPS VI-4.2.3 External Libraries

Clear

SPS VI-4.2.3.1 Current Status Notebook

Clear

SPS VI-4.2.3.2 Archives

Clear

SPS VI-4.2.3.3 Run Notebooks

Clear

SPS VI-4.3 Program Organization

Clear

SPS VI-4.3.1 Specification Format

Clear

SPS VI Section 5 Program Specification - Batch Environment

Not reviewed

SPS VI Section 6 Program Specification - Terminal Environment

SPS VI-6.0 On-Line Control Subsystem (OCS)

Clear

The implementation of an on-line control subsystem (OCS) in the manner stated will be detrimental to the expeditious use of the PSL. As stated in paragraph 6.0 of Volume VI, each time the user enters the PSL and each time he types a command he will be bombarded with menus. Some recognition should be given to the programmer who has already shown the capability to converse intelligently in multilingual environments be it ENGLISH, FORTRAN, COBOL or JOVIAL. To accomplish this it is recommended that, upon being entered, the PSL respond with a prompting identifier such as PSL:. At this point any command including HELP may be entered. When HELP is typed, a menu which is limited to the current level of entry into the command structure will be received. If the user finds himself too deep in the command structure, he should be able to type a command that will automatically back up one level such as UP. HELP may be requested at this level for a menu. Three levels of menu display should be available. The first is for the sophisticated user who merely wants to be reminded of the commands. This would be invoked by typing HELPS, for "help short". When HELPS is typed only the command names will be displayed. The second level is for the less sophisticated. He would type HELP and receive the list of commands and the functions they perform in a brief description. The third level of aid available would be a detailed description of the command capability possibly including examples or reference to a users manual. This would be invoked by typing "HELP, command" where the command in question is substituted into the string.

The small contractor was able to accomplish many of the functions required of the PSL by using existing features of operating systems. In the review and analysis of Volume VI it will be assumed that the small contractor will be able to use these features and needs to implement PSL requirements only when they are clearly not available.

SPS VI-6.1 Library System Maintenance (LSM)

SPS VI-6.1.1 PSL Installation (INSTALL)

The small contractor would not need the install feature. Its main function is to start a PSL catalog. The PSL catalog is used to separate projects that are active. The small contractor usually finds it easier to separate each project either by a unique volume on the in-house system, or a unique userid on the commercial time sharing system or the Government computer.

SPS VI-6.1.2 Project Initialization (INITIAL)

Initiating a project catalog can be achieved by the small contractor without the command INITIAL. Since each project has its own volume or userid the system will automatically record the units entered into the library as individual files in the directory of the volume, and the userid can be considered as one complete volume.

SPS VI-6.1.3 File Allocation (ALLOCATE)

When the small contractor plans for a project he manually allocates a storage area for the computer image data. This is either a volume on the in-house, or a userid on the commercial time sharing or Government computer. With initial allocation completed, reallocation of direct access storage is accomplished by the operating system. When a file is updated, usually via the editor, the old file space is returned to the system for future use. The actual location on the random access device (RAD) of the file is retained by the system. The small contractor may access this information but does not normally do so. The RAD directory is automatically updated by the system simulating the use of a project catalog. In its default mode the file accounting record sets all flags off. The small contractor will have no problem using the default mode. Each data item of the file accounting record will be discussed in its non-default mode individually. The library file password capability is not available on the in-house system. When using the commercial time sharing system, the default actually changes to yes, and a password is required. The Government computer system allows passwords on each unit of data so that a common password may be issued for an entire library if desired.

The security classification must remain the default of "none" on the commercial time sharing system. When using the in-house system, it is possible to maintain security levels. Only those individuals concerned with the project would be able to access the library file storage volume from the computer room so that security would be maintained. Under special circumstances the Government computer could be accessed for classified work. The situation would be similar to the in-house closed computer room where all secure information is retained within the realms of the project.

When Volume V was reviewed, it was shown how data compression was accomplished as an option on the commercial time sharing system. When stored in compressed mode, the file had a unique identifier. It was recommended at that time to create a data compression/restoration routine for use by PSL systems. It was also recommended that the default mode be "compressed" and that modifications be made to all routines that access the file to allow them to do so in the compressed mode.

Neither automatic nor non-automatic SP checking is available to the small contractor. It is assumed that routines will be available to the small contractor to perform non-automatic checking for each higher-order language on both the Government computer and the commercial time sharing system. Since the small contractor will be invoking an on-line editor it is recommended that the SP check be an optional exit mode from the editor. The other computing centers accessed by the small contractor might also offer this automatic SP check mode. The non-default mode of automatic collection of management data at the unit level is not completely available to the small contractor. The unit start data is available on all three systems. The date last modified is available only on the commercial time sharing system and the Government computer. The time last modified is available only on the Government computer. Assuming that the small contractor employs a milestone technique to ascertain the level of project completion, then the collection of data for the expanded version of the UAR would not have to be carried out.

A unit lines flag would have to be implemented since none is presently available. One possible method of achieving the result would be a modification of the editor to automatically check the line count when exiting.

SPS VI-6.1.4 File/Library/Project Termination (TERMINATE)

The terminate subprogram as described in Volume VI, subsection 6.1.4, is not needed by the small contractor. On the in-house system a delete command is available to remove a file from a volume. The commercial time sharing system provides the ERASE command while the Government computer has a PURGE command. If it is desired to backup a file before removing it from the system, the file or files may be copied to magnetic tape as has been shown in the examples presented in the discussion of Volume V above.

SPS VI-6.1.5 Authorized Project User List Maintenance (NTABLE)

The small contractor using an in-house system does not have the capability to use an NTABLE. The need for such a file is open to question due to the relatively small number of people actually involved with the project. When accessing a commercial time sharing system, the project userid which contains a library is protected by a password. If additional protection is required, a PROFILE EXEC may be implemented. This is a job that is executed each time the operating system is invoked. It has the capability to ask for and verify additional passwords. If the password is not correctly entered, then the user can be logged off the system. Figure II.3.2-92 shows a sample PROFILE EXEC that prompts for an additional password and checks it against a list of acceptable character strings.

If the userid is attachable then a file called PROTECT EXEC can be created similar to the PROFILE EXEC. The PROTECT EXEC is executed each time the userid is attached. Instead of logging out the user when an incorrect password is entered, the PROTECT EXEC should detach itself. When using the Government system, it is possible to include a password on each unit of a library. By controlling the dissemination of the passwords, it would be possible to control access to the files.

```
14.52.32>PRINTF PROFILE EXEC

&TYPE OFF
&PRINT PASSWORD
&PRINT + XXXX
&READ ARGS
&IF &INDEX EQ 0 &GOTO -ADD
&IF &1 EQ KEY &QUIT
-ADD &INDEX + 1
&IF &INDEX1 EQ 3 LOGOUT
&SKIP -7

14.54.48>PROFILE

PASSWORD
 XXXX

14.55.39>PROFILE

PASSWORD
 XXXX
PASSWORD
 XXXX
13.010  VPU'S, 0.34 CONNECT HRS, 271 I/O
LOGGED OFF AT 14.58.23 ON 15 NOV 76
```

Figure II.3.2-92. Simulating NTABLE on a commercial time sharing system by use of PROFILE EXEC.

SPS VI-6.2 Source Data Maintenance (SDM)

SPS VI-6.2.1 File Update (UPDATE)

The UPDATE feature of SDM described in Volume VI, subsection 6.2.1 is similar to a crude editor. When Volume V was reviewed, detailed examples were given of the EDIT capabilities on the three systems. The editors covered are sophisticated routines that accomplish all of the described tasks of UPDATE with the exception of automatically updating the UAR and printing a new listing of the unit. It is recommended that the UPDATE feature be renamed to EDITOR for the OCS and that any size contractor using an OCS be allowed to substitute the system editor. If any of the features of

UPDATE are not available to the OCS editor, they could be implemented. One feature that is not desirable is the automatic printing of the updated unit especially if it is a source unit. This small contractor has found that compiling and test running the file before printing often identifies any errors, such as typographical, that have been entered into the file and may be corrected before the file is printed. This results in decreased material costs.

SPS VI-6.2.2 Merge Two Library Files (MERGE)

The MERGing of two library files can be accomplished by the small contractor on all three computers. This may not be the optimum method of accessing a file from two PSL systems. For example, if the user of the commercial time sharing system has more than one disk attached to his virtual machine then most commands to the system will automatically search all of his disks for a specified unit. This allows one copy of the file to exist which results in less storage being used. An additional feature is that only one file need be changed if modifications are necessary. If unique modifications must be made for a particular project then a copy has to be generated.

SPS VI-6.2.3 File/Library/Project Backup (BACKUP)

BACKUP facilities were described in the discussion of Volume V. In the examples presented, single units were transferred from disk to magnetic tape. The option to BACKUP entire volumes with one command is available on both the in-house system and the commercial time sharing system. This is accomplished by the so called "wild card" construct. Instead of entering the file identifiers, the asterisk (*) is entered in their place. Using the "wild card" option in both file identifiers will BACKUP all files from the disk onto the magnetic tape. The typical Government computer does not have the capability of "wild card" constructs. In order to accomplish a BACKUP of the entire library, it is necessary to identify each unit of the library.

SPS VI-6.2.4 Unit/File/Library/Project Restoration (RESTORE)

To RESTORE a file that has been backed up the small contractor can invoke the system utilities in existence. One feature of the RESTORE subprogram that is of questionable value is the fact that it does not have the ability to overwrite an existing disk-resident version of the file. The user must specifically TERMINATE and re-ALLOCATE the

file before it can be RESTOREd. Both the in-house system and the commercial time sharing system automatically overwrite the existing copy of the file, thus saving invocation of two PSL commands. The Government system would attempt to add a cycle of the file identified and retain the replaced version if possible. If the wild card, also referred to as global construct, is not utilized, then any file or unit may be selected.

SPS VI-6.3 Output Processing (OUTPUT)

SPS VI-6.3.1 File Directory Index Output (INDEX)

The INDEX subprogram is available in various levels of implementation to the small contractor depending upon which system is being accessed. The in-house system has the lowest level of implementation while both the commercial time sharing system and the Government computer have full INDEX capabilities. Missing from the in-house system are the project/library name, the time, page numbering and volume ID. It is believed that these could be added quite easily to the utility program to provide full INDEX capabilities.

SPS VI-6.3.2 Source Data Output (SOURCE)

The capabilities of the SOURCE subprogram of the SDM are generally available to the small contractor by various commands that are functions of the systems. The production of a classification title at the top and bottom of the printed page needs to be implemented on the in-house system but would not be applicable to the commercial (or Government) computers. Printing of compressed files could be implemented in-house if a compress/restore feature were added to the system. Automatic indentation of the file to SP standards should not be necessary since the file should be stored in SP format. The file, when compressed, will not appreciably add to the amount of storage requred even if it is stored in SP indented format. Either a single unit of data or an entire library may be printed. Files or units may be copied onto magnetic tape by commands already available in the system. Punched card output is available on either the commercial time sharing system or the Government computer. The in-house system of the small contractor might have a card punch but with industry compatible magnetic tape units this small contractor has not found a need for paper-card-processing peripherals. It should be noted that a card reader was specified in Table 3-1 of Volume VI as a minimum equipment requirement. However, a card punch was not specified. On large computer systems, the peripheral for punched card manipulation often has both reading and punching capabilities which might be the reason a

punch was not specified. The small contractor is more likely to have a minicomputer in-house for which he can purchase the card reader and card punch as two separate peripherals. The advantages of punched card output are debatable in the on-line environment used by the small contractor. The large contractor might also consider the release of his computer from the slow card devices since magnetic media are available to replace them. The reader is directed to a case study of a batch oriented user who converted to a cardless operation and reported significant savings (see "Going Cardless" in DATAMATION magazine of September 1976).

Displaying a unit of data is possible on all of the systems already. Depending upon how the unit is accessed, any portion of the unit may be displayed or the entire unit or even the entire library.

SPS VI-6.3.3 Programmer's Directory/Listings Output (AUTHOR)

The AUTHOR subprogram is not available to the small contractor on any of the systems he normally accesses. It is recommended that an AUTHOR type subprogram be developed for the small contractor. It is a common practice to include a comment line in a source code file that identifies the author of the code. It is suggested that the author subprogram be written such that it searches all source units of a library for this comment line rather than using a UAR.

SPS VI-6.3.4 Character String Scan (CSCAN)

If the AUTHOR subprogram is implemented as recommended above, then the CSCAN subprogram will be a slightly modified version of it. The in-house system supports scanning for a string of characters in a file via the editor. The commercial time sharing system can also use the editor to search for a character string or if an entire library needs to be searched an EXEC program could be set up to scan all of the files for the string. The Government computer does not have EXEC capabilities but the editor is accessable from a batch job which can execute a stored instruction file.

SPS VI-6.4 Management Data Collection and Reporting (MDCR)

SPS VI-6.4.1 Management Data File Initialization/ Termination (MDFILE)

The small contractor employing a milestone-achieved reporting technique may not need the MDFILE subprogram for the

MDCR feature of the PSL.

SPS VI-6.4.2 Management Data Collection/Update (MDUPDATE)

MDUPDATE can be accomplished with the edit feature of the operating systems accessed by the small contractor.

SPS VI-6.4.3 Establish Automatic Management Reporting (MDREPORT)

The purpose of the MDREPORT subprogram is to enter report start date, cycle and required reports into a table. The small contractor can invoke the editor to generate the appropriate table.

SPS VI-6.4.4 Management Data Report Printing and Archiving (MDHIST)

MDHIST is a subprogram to scan magnetic tape and produce a summary of accomplishments from reports produced by MDPRINT. MDPRINT archived only the requested reports. If MDHIST requires other information, it will not be available from the tape. The small contractor may not utilize MDPRINT to generate reports and therfore would not need MDHIST.

SPS VI-6.4.6 Establish Automatic Exception Checking (MDXCHECK)

The description of MDXCHECK is incomplete. The standard should provide a tentative list of unique mnemonics or identifiers for allowable input items that can have variances. A fuller discussion of what items could be tracked and the purpose of tracking them is presented in SPS Volume IX. Management reporting techniques employed by this small contractor already allow for a close scrutiny of project variance beyond allowed limits which will be further discussed when Volume IX is reviewed.

SPS VI-6.4.7 User Routine Support (MDUSER)

The small contractor has the editor available to modify a file. This implements the MDUSER subprogram.

SPS VI-6.5 Programming Language Processor (PLP)

SPS VI-6.5.1 Precompile/Compile Interface (COMPILE)

The COMPILE subprogram, as described, invokes three routines which are

1. an INCLUDE resolver,

2. a precompiler,

3. a language compiler.

Expected output is a precompiler-generated listing, a language-compiler generated listing and, optionally, an object module. It was stated in SPS Volume I, subsection 5.4.1 that "the programmer-written source code listing representing input to the precompiler should be used for debugging purposes instead of the compiler output listing". Following this premise, presented in the basic standard, the language compiler output listing should not be automatically generated but only available to the programmer on an exception basis. Omission of the compiler listing will decrease the size of the OBJECT notebook by up to 50 percent. Also a decreased material cost will be realized. It is recommended that the conflicting statements in SPS Volumes I and VI in regard to the generation of the compiler listing be resolved and that the premise presented in SPS Volume I, subsection 5.4.1 be followed as opposed to SPS Volume VI, subsection 5.5.1, paragraph 4. The small contractor cannot execute a series of programs on line utilizing his in-house computer. It is recommended that a facility similar to the EXEC capability on the commercial time sharing system be implemented for the small contractor's in-house computer. It is also recommended that a routine to resolve INCLUDEs be developed for the small contractor. The commercial time sharing system EXEC feature allows the programmer to store a sequence of commands to be executed. Conditional testing and branching are also a capability of the EXEC feature. The small contractor could set up an EXEC file on the commercial time sharing system to implement the COMPILE command of the PLP which is part of the PSL. All interfaces to the operating system are available. The command sequence would have to be generated and the EXEC file would be available. The Government computer system has stored command sequence execution capabilities in batch mode which the user can activate from his on-line terminal. It is not a direct method but it would accomplish the COMPILE subprogram's stated tasks. It should be noted that EXEC on the commercial time sharing system can accept execution parameters from the user but the batch facility of the Government computer cannot. On the Government computer a master batch COMPILE file could be available to the user. He would then make appropriate substitutions to the master file before having batch execute the job stream. Due to the ability of the EXEC file

to execute all system commands and to test specific conditions, it could direct the resolve INCLUDE program to the appropriate libraries and invoke the appropriate precompiler and compiler.

It is recommended that an added optional feature be added to COMPILE in the form of a GO option that would allow the user to compile, load and execute the program.

SPS VI-6.5.2 Linkage Interface (LINK)

The generation of program load modules by LINKING object modules is a system function. The added convenience of storing the names of the files to LINK could be implemented in-house by adding the EXEC capability which exists on the commercial time sharing system. The full function of the LINK command is to link, save and go. One of the stated capabilities of LINK is to generate dummy modules for unsatisfied references. This is a dangerous technique. It may generate modules for code that the user has generated but might have omitted from the input list. Also stubs would have been generated in previous steps so that no dummy modules should need to be created. It is recommended that the ability to generate dummy modules by the LINK subprogram be removed.

SPS VI-6.5.3 Execution Interface (EXECUTE)

The EXECUTE function is available to the small contractor accessing an on line system. The command names vary on the three example systems but each is capable of loading a program module and starting its execution.

SPS VI-6.6 Documentation Processing (DOCUMENT)

Documentation processing is available to the small contractor already so he would not need a specific command to DOCUMENT his on-line files. The standard suggests that a feature be available to process textual files to a listing device with optional directives for formatting the output. There are programs available to do this. The commercial time sharing system has a SCRIPT capability to produce printed reports from textual files. It is optionally available to direct the output which normally goes to the line printer or terminal to a magnetic tape.

SPS VI-6.7 Additional Terminal Functions

SPS VI-6.7.1 Authorized Terminal/Project List Maintenance (TTABLE)

The TTABLE subprogram for OCS is not necessary. The operating system provides file security. This small contractor

has experience with many on line systems. The larger systems require the user to know a userid and its associated password, as a minimum, to access the system. Regular modification of the password and sometimes even the userid prove to be effective means of retaining the security of the files on the logins. Additional security can be achieved by the utilization of special files, the PROFILE and PROTECT EXECs previously mentioned for the commercial time sharing system and the addition of passwords to the individual files on the Government computer. Multiuser minicomputer operating systems often used by the small contractor also offer file security by passwords.

SPS VI-6.7.2 Message Processing (MESSAGE)

MESSAGE processing should not be required by the small contractor since the number of people involved with the project is relatively small and the physical distance between the people would not be as great as might occur for large contractors whose project personnel might be spread throughout an entire office building. The in-house system does not support MESSAGE processing. However, the commercial and Government computer systems do. Both systems also offer the user the option to reject messages, that is to stop receipt of any messages except a high priority message sent by the computer operator.

SPS VI-6.7.3 Terminal Statistics Reporting (TSR)

TSR reporting of the PSL access time is automatically provided by the accounting facilities of the commercial time sharing system and the Government computer. Depending upon the accounting algorithm used, various other parameters are measured. Periodic reports are sent to the user to bill him for the facilities accessed. The commercial time sharing system also has the capability to automatically log this information to a user's disk so that he may access it. If this option is not exploited a special EXEC file may be implemented on the commercial time sharing system to record the terminal usage upon logout. Many of the statistics suggested to be maintained by OCS were of little value to the small contractor who finds it more meaningful to measure the level of project completion by milestones achieved rather than by the number of times a file is accessed.

SPS VI-6.8 OCS HIPO diagrams

Clear

SPS VI Section 7 System Generation

SPS VI-7.1 PSL Generation

Clear

SPS VI-7.2 Data Protection Scheme

Clear

SPS X-A.2.3 CPT Administrative Assistant

Clear

SPS X Appendix B Structured Programming Technology Fundamentals

SPS X-B.1 Introduction

Clear

SPS X-B.2 TDSP

Clear

SPS X-B.2.1 SP

Clear

SPS X-B.2.2 Use of Subroutines

Clear

SPS X-B.2.3 Top Down Programming

Clear

SPS X-B.3 PSL

Clear

SPS X-B.4 PDL

Clear

SPS X-B.5 Definitions

Clear

SPS XV Validation and Verification Study

SPS XV Section 1 Introduction

SPS XV-1.1 Background

Clear

SPS XV-1.2 Report Organization

Clear

SPS XV-1.3 Conclusions

Clear

SPS XV-1.4 Recommendations

Clear

SPS XV Section 2 Definition Framework for Verification and Validation

SPS XV-2.1 Terminology History

Clear

SPS XV-2.2 Definitional Framework

Clear

SPS XV-2.2.1 Verification Definition

Clear

SPS XV-2.2.2 Validation Definition

Clear

SPS XV-2.3 Related Testing Activities

SPS XV-2.3.1 Correctness Proof

Clear

SPS XV-2.3.2 Certification

Clear

SPS XV-2.3.3 Performance Testing

Clear

SPS XV Section 3 V/V Techniques Survey

SPS XV-3.1 Introduction

Clear

SPS XV-3.1.1 Computer Based Testing

Clear

SPS XV-3.1.2 Manual Based Testing

Clear

SPS XV-3.2 Verification Techniques

SPS XV-3.2.1 Drivers

SPS XV-3.2.1.1 Description

Clear

SPS XV-3.2.1.2 Use

Clear

SPS XV-3.2.1.3 Advantages

Clear

SPS XV-3.2.1.4 Disadvantages

Clear

SPS XV-3.2.2 Test Data Bases

SPS XV-3.2.2.1 Description

Clear

SPS XV-3.2.2.2 Use

Clear

SPS XV-3.2.2.3 Advantages

Clear

SPS XV-3.2.3.4 Disadvantages

Clear

SPS XV-3.2.3 Design Verification

SPS XV-3.2.3.1 Description

Clear

SPS XV-3.2.3.2 Use

Clear

SPS XV-3.2.3.3 Advantages

Clear

SPS XV-3.2.3.4 Disadvantages

Clear

SPS XV-3.2.4 Execution Analysis

SPS XV-3.2.4.1 Description

Clear

SPS XV-3.2.4.1.1 Program Evaluator and Tester (PET)

Clear

SPS XV-3.2.4.1.2 Product Assurance Confidence Evaluator (PACE)

Clear

SPS XV-3.2.4.1.3 Program Testing Translator (PTT)

Clear

SPS XV-3.2.4.1.4 Software Implementation Monitor (SIMON)

Clear

SPS XV-3.2.4.2 Use

Clear

SPS XV-3.2.4.3 Advantages

Clear

SPS XV-3.2.4.4 Disadvantages

Clear

SPS XV-3.2.5 Automated Network and Path Analysis

SPS XV-3.2.5.1 Description

Clear

SPS XV-3.2.5.2 Use

Clear

SPS XV-3.2.5.3 Advantages

Clear

SPS XV-3.2.5.4 Disadvantages

Clear

SPS XV-3.2.6 Statistical Prediction

SPS XV-3.2.6.1 Description

Clear

SPS XV-3.2.6.2 Use

Clear

SPS XV-3.2.6.3 Advantages

Clear

SPS XV-3.2.6.4 Disadvantages

Clear

SPS XV-3.3 Validation Techniques

SPS XV-3.3.1 Functional Testing

SPS XV-3.3.1.1 Description

Clear

SPS XV-3.3.1.2 Use

Clear

SPS XV-3.3.1.3 Advantages

Clear

SPS XV-3.3.1.4 Disadvantages

Clear

SPS XV-3.3.2 Design Simulation

SPS XV-3.3.2.1 Description

Clear

SPS XV-3.3.2.2 Use

Clear

SPS XV-3.3.2.3 Advantages

Clear

SPS XV-3.3.2.4 Disadvantages

Clear

SPS XV-3.3.3 Design Validation

SPS XV-3.3.3.1 Description

Clear

SPS XV-3.3.3.2 Use

Clear

SPS XV-3.3.3.3 Advantages

Clear

SPS XV-3.3.3.4 Disadvantages

Clear

SPS XV-3.3.4 Matrix Analysis and Problem Statement Language

SPS XV-3.3.4.1 Description

Clear

SPS XV-3.3.4.2 Use

Clear

SPS XV-3.3.4.3 Advantages

Clear

SPS XV-3.3.4.4 Disadvantages

Clear

SPS XV-3.4 Summary

Clear

SPS XV Section 4 Structured Programming Project Survey

SPS XV-4.1 Introduction

Clear

SPS XV-4.2 Mitsubishi Bank New System - Japan

SPS XV-4.2.1 Project Overview

Clear

SPS XV-4.2.2 SPT Components

Clear

SPS XV-4.2.3 Test Environment

Clear

SPS XV-4.2.4 V/V

Clear

SPS XV-4.3 Ground Support Simulation Computer for ASTP - Houston, Texas

SPS XV-4.3.1 Project Overview

Clear

SPS XV-4.3.2 SPT Components

Clear

SPS XV-4.3.3 Test Environment

Clear

SPS XV-4.3.4 V/V

Clear

SPS XV-4.4 Advanced Control System (ACS) - Houston, Texas

SPS XV-4.4.1 Project Overview

Clear

SPS XV-4 4.2 SPT Components

Clear

SPS XV-4.4.3 Test Environment

Clear

SPS XV-4.4.4 V/V

Clear

SPS XV-4.5 Earth Resources Interaction Processing System (ERIPS) - Houston, Texas

SPS XV-4.5.1 Project Overview

Clear

SPS XV-4.5.2 SPT Components

Clear

SPS XV-4.5.3 Test Environment

Clear

SPS XV-4.5.4 V/V

Clear

SPS XV-4.6 FAA National Air Space System Support Software (NASCOR, ACEUTE) - Atlantic City, New Jersey

SPS XV-4.6.1 Project Overview

Clear

SPS XV-4.6.2 SPT Components

Clear

SPS XV-4.6.3 Test Environment

Clear

SPS XV-4.6.4 V/V

Clear

SPS XV-4.7 Resource Monitoring System Force Status - Rosslyn, VA

SPS XV-4.7.1 Project Overview

Clear

SPS XV-4.7.2 SPT Components

Clear

SPS XV-4.7.3 Test Environment

Clear

SPS XV-4.7.4 V/V

Clear

SPS XV-4.8 Multi-Optical Sensor Simulation (MOSS) - Westlake, CA

SPS XV-4.8.1 Project Overview

Clear

SPS XV-4.8.2 SPT Components

Clear

SPS XV-4.8.3 Test Environment

Clear

SPS XV-4.8.4 V/V

Clear

SPS XV-4.9 Test Control Computer Support Software (TCCSS) - Gaithersburg, MD; Endicott, NY

SPS XV-4.9.1 Project Overview

Clear

SPS XV-4.9.2 SPT Components

Clear

SPS XV-4.9.3 Test Environment

Clear

SPS XV-4.9.4 V/V

Clear

SPS XV-4.10 Trident AN/BQQ-6 Sonar Software - Manassas, VA

SPS XV-4.10.1 Project Overview

Clear

SPS XV-4.10.2 SPT Components

Clear

SPS XV-4.10.3 Test Environment

Clear

SPS XV-4.10.4 V/V

Clear

SPS XV-4.11 Energy Management System (EMS) - Houston, TX

SPS XV-4.11.1 Project Overview

Clear

SPS XV-4.11.2 SPT Components

Clear

SPS XV-4.11.3 Test Environment

Clear

SPS XV-4.11.4 V/V

Clear

SPS XV Section 5 SPT Impact on V/V

SPS XV-5.1 Introduction

Clear

SPS XV-5.2 Techniques Associated With SPT

SPS XV-5.2.1 Top Down Programming

Clear

SPS XV-5.2.2 Use

Clear

SPS XV-5.2.3 Advantages

Clear

SPS XV-5.2.4 Disadvantages

Clear

SPS XV-5.2.2.5 SPT Impact on Code Verification

Clear

SPS XV-5.3 SPT Impact on Techniques Described in Section 3

Clear

SPS XV-5.3.1 Drivers

Clear

SPS XV-5.3.2 Design Verification

Clear

SPS XV-5.3.3 Execution Analysis

Clear

SPS XV-5.3.4 Automated Network and Path Analysis

Clear

SPS XV-5.3.5 Functional Testing

Clear

SPS XV-5.3.6 Design Simulation

Clear

SPS XV Appendix A Software Development Cycle

SPS XV-A.1 Introduction

Clear

SPS XV-A.2 Development Phase

SPS XV-A.2.1 System Definition

Clear

SPS XV-A.2.2 System Design

Clear

SPS XV-A.2.3 System Implementation

Clear

SPS XV-A.2.4 System Evaluation

Clear

SPS XV-A.2.5 Relationship of Phases

Clear

SPS XV Appendix B SPT Fundamentals

SPS XV-B.1 Introduction

Clear

SPS XV-B.2 TDSP

Clear

SPS XV-B.2.1 SP

Clear

SPS XV-B.2.2 Use of Subroutines

Clear

SPS XV-B.2.3 TDP

Clear

SPS XV-B.3 PSL

Clear

SPS XV-B.4 PDL

Clear

SPS XV-B.5 Definitions

Clear

II.3.3 Applicability of Structured Programming Standards and Guidelines

The subject of the applicability of Structured Programming Standards and Guidelines is addressed in Volumes I, Programming Language Standards, and IV, Data Structuring Study of the Structured Programming Series. This Chapter II.3.3 provides detailed comments, examples and recommendations concerning any chapter within the Structured Programming Series where the subject of the Programming Standards and Guidelines are mentioned.

SPS I-3.2 Standards

a) Clear.

b) Compliance with Standard 2, "Explicit branching (GOTO type instructions) is not permitted" would be difficult to achieve.

c) A problem is foreseen in implementing the structure IFTHENELSE. For example, in FORTRAN to implement IF A EQ D THEN A=B ELSE A=C, it would follow to use
IF(A.EQ.D) A=B
A=C,
but this will set A=C always.
By using a GOTO it could be stated

```
      IF(A.EQ.D) GOTO 1000
      A=C
      GOTO 1100
1000  A=B
1100  ...
```

This maintains the IFTHENELSE structure.

d) We would recommend that this standard be removed. A guideline should replace it. The guideline should be stated "Explicit branching (GOTO type instructions) is permitted only in the forward direction". The inclusion of the condition maintains the flow from top to bottom of the listing. Continuity and ease of reading will be retained.

SPS I-3.3 Guidelines

a) Clear. The use of guidelines for a project has always improved the performance on a contract. Basic rules for assisting the programmer in accomplishing the task in a uniform manner are used extensively by small contractors.

SPS I-4.2.2.1 TDSP in Jovial J73 and J73/1 IFTHENELSE

a) clear
b) recommend labeling "ENDIF"

SPS I-4.2.2.2.1 TDSP in Jovial J73 and J73/1 DOWHILE

a) clear

SPS I-4.2.2.2.2 TDSP in Jovial J73 and J73/1 DOWNTIL

a) clear, but exceptions are getting extensive
d) recommend labeling "DO UNTIL (P) and "ENDDO"
e) Example

```
"DO UNTIL (P) D1010

NAME:  Code A
       IF NOT (P); GOTO NAME;
     "ENDDO D1010"
```

with labels, the beginning and end of the loop are clearly identified, this adds to the readability. Indentation is an attempt to add clarity but due to the possible length of code A a label would make the loop stand out. Also if nested looping were used then the label would help even more in promoting readability.

Example

```
CODE A
WHILE (P);
   BEGIN
     CODE A;
     CODE B;
     WHILE (q):
       BEGIN
         CODE B;
     END;
     "ENDWHILE"
   END;
"ENDWHILE"
```

or

```
"DOUNTIL DU0120"
CODE A
WHILE (P);
```

```
   BEGIN
     CODE A;
     "DOUNTIL DU0110"
     CODE B;
     WHILE (q)
       BEGIN
       CODE B;
       END
     "ENDWHILE DU0110"

   END;
"ENDWHILE DU0120"
```

SPS I-4.2.2.3 TDSP Jovial J73 and J73/1 Case figure

a) clear

d) recommend that indentation be greater, at least length of number. The following 2 examples show the guideline indentation in the first and the recommended indentation in the second.

Example 1

```
SWITCH numeric:  formula;

  [number]
    code A,
  [number]
    code B,
  [number]
    code C,
  :
ENDSWITCH
```

Example 2

```
SWITCH numeric:  formula;

[number]
        code A
[number]
        code B
[number]
        code C
:
ENDSWITCH
```

Even this simple example shows that the transfer location is readily discernable from the code portions of the program. This would be especially beneficial when the code portions are greater than a few statements.

SPS I-4.2.2.4 TDSP in Jovial J73 and J73/1 Include

a) Clear

SPS I-4.2.3.1 Restricted Jovial J73 Statement Usage

a) Clear

SPS I-4.2.3.2 Program Organization

a) Clear

SPS I-4.2.3.3 Comments

a) Clear

d) We recommend that comments be included to ensure maintainability of the code. By including them in the listing they are more likely to remain with the program and the PSL will not have to be searched to obtain them.

SPS I-4.2.3.4 Identifying and Formatting Conventions

a) Clear

SPS I-4.3.2.1 JOVIAL J3 IFTHENELSE

a) Clear

d) Recommend the inclusion of comment "ENDIF"

SPS I-4.3.2.2.1 Jovial J3, DOWHILE

a) Clear, the complexity of utilizing flags overrides its clarity

SPS I-4.3.2.2.2 Jovial J3, DOUNTIL

a) Clear

SPS I-4.3.2.2.3 Compound DO Loops

a) Clear

SPS I-4.3.2.3 Case Figure

a) Clear

SPS I-4.3.2.4 Include

a) Clear

SPS I-4.3.3.1 Restrictions

a) Clear

SPS I-4.3.3.2 Program Organization

a) Clear

SPS I-4.3.3.3 Comments

a) Clear

SPS I-4.3.3.4 Indentation and Format Conventions

a) Clear

SPS I-4.4.1 FORTRAN General Comments

a) Clear

The General Comments concerning FORTRAN are generally negative, which is the common attitude among computer scientists and vendors promoting other languages which run only on a limited number of computers. However, since FORTRAN is the most widely used language for application programming and also creates the least problems in transferring programs from one type of computer to another, viable techniques will have to be established to allow Structured Programming in FORTRAN.

SPS I-4.4.2 TDSP in FORTRAN

a) Clear

SPS I-4.4.2.1 IFTHENELSE

a) Clear

With the inclusion of the boolean .NOT. in the statement the complexity is increased which runs opposite to the goals of structured programming. Use of the relational operators (.NE., .GT., .LT., .LE.) allowed in the logical FORTRAN IF would decrease the ungainliness of the statement and clearly denote to the reader the intent of the tested condition which will reinforce the comment ENDDO terminating the loop.

SPS I-4.4.2.3 Case Figure

a) Clear

SPS I-4.4.2.4 INCLUDE Capability

a) Clear, but a cumbersome solution is presented.

b) Compliance with the standard to use the "INCLUDE" capability would present no problem since it is generally already implemented on a sophisticated commercial time sharing system. See Figure II.3.3-1 which is an excerpt from an international time sharing service FORTRAN users manual.

The INCLUDE feature is not available on smaller in-house computer systems or at Government installations.

INCLUDE CARD

Frequently in a group of programs and subprograms, a series of statements are identical in each program. Specification statements are typical – such as DIMENSION, COMMON, EQUIVALENCE, etc. Groups of executable statements may also be duplicated among programs.

A feature of the FORTRAN compiler, the INCLUDE card is also a feature of the HFORTEX compiler. It allows one or more statements to be "pulled in" from a MACLIB type file, instead of appearing explicitly in each program. This feature can be used to significantly reduce the physical size of source programs that are modularized. It also allows rapid change of items that are common to many programs (e.g., COMMON) without having to change all the affected programs.

In order to use the INCLUDE feature, one or more sequences of legitimate FORTRAN statements are placed in one MACLIB type file (using the MACLIB command), and one or more MACLIB files are specified (using the GLOBAL command) for automatic searching during compilation.

Each separately named sequence of statements in a MACLIB is called a member. When an INCLUDE card is encountered, every GLOBAL MACLIB file is searched for a member with the appropriate name. When found, all of the statements in that member are copied into the program being compiled replacing the INCLUDE card.

INCLUDE cards are written:

INCLbbmembernm (b indicates a blank character)

where INCL begins in column 1, membernm begins in column 7 and membernm is any 1 to 8 character name. MACLIB type files to be used with FORTRAN are created and used as follows:

1. Create each member as a separate file with filetype of COPY (or with filetype FORTRAN and then alter to the filetype COPY).

2. Use the MACLIB GEN command to generate a MACLIB type file from the COPY files containing the individual members. Members can also be added or deleted using the MACLIB command. (See the *VP/CSS Reference Manual*, Form 106, for details about the MACLIB command.)

3. Before compilation of a program that contains INCL cards, specify the MACLIB file(s) to be searched using the GLOBAL MACLIB command.

4. Compile the program(s).

Figure II.3.3-1. Description of the INCLUDE capability of a commercial time sharing system.

EXAMPLE

```
xx.xx.xx edit alpha copy as fortran
NEW FILE.
INPUT:
common /bill/ abc, def
common /jane/ ijk (12), lmz, jkj
EDIT:
file

xx.xx.xx maclib gen hcopy alpha

xx.xx.xx printf gamma fortran

      BLOCK DATA
INCL  ALPHA
      DATA IJK/12*0/
      DATA ABC/15.0/
      END

xx.xx.xx global maclib hcopy

xx.xx.xx hfortex gamma (opt 1)

xx.xx.xx
```

NOTES

a. No special compiler options need be specified to use this facility.

b. The INCLude facility is not recursive.

ERROR MESSAGES

1. MACLIBname NOT OPENED. . .09

   A MACLIB file specified in the GLOBAL command was not found on disk. Successful compilation of the program is possible only if all members to be INCLuded are available from other MACLIBs.

2. In the example above the following messages would be generated and the compilation of that particular program would terminate if ALPHA was not found in HCOPY:

   MEMBER ALPHA NOT FOUND IN AVAILABLE LIBRARIES.

NUMBER	LEVEL	FORTRAN H EXTENDED ERROR MESSAGES
IFE401I	16(U)	COMPILATION DELETED. A MESSAGE FOR AN UNRECOVERABLE ERROR HAS BEEN ISSUED.

Figure II.3.3-1 (continued). Description of the INCLUDE capability of a commercial time sharing system.

d) It is recommended that the Government make this feature available on their systems.

SPS I-4.4.2.2.1 DOWHILE

a) Clear, however a clearer implementation of the DOWHILE segment would be

```
C     DOWHILE I.LE.K  DW0010
      IF (J.LT.K) GOTO 0030
      DO  0010 I=J,K,L
      CODE A
 0010 CONTINUE

C     END DOWHILE DW0010
 0030 CONTINUE
```

In this method the compiler takes care of the index of the DO. It is not left up to the programmer to ensure the incrementation of the index in Code A, which is a possible source of error.

SPS I-4.4.2.2.2 DOUNTIL

a) Clear - The same comments that apply to DOWHILE apply here.

SPS I-4.4.2.2.3 The FORTRAN DO

a) Clear.

Omission of the CONTINUE statement will detract from the clarity of the program. Including the CONTINUE in structured programs will aid readability.

SPS I-4.4.3.1 Restricted FORTRAN Statement Usage

a) Clear

b) Considering that the use of GOTO statement is supposedly forbidden in Structured Programming, it is mildly amusing to see how often it is actually employed to implement structured FORTRAN.

SPS I-4.4.3.2 Program Organization

a) Clear

SPS I-4.4.3.3 Comments

a) Clear

SPS I-4.4.3.4 Statement Numbering

a) Clear

d) Recommend: It is advisable to use ranges of numbers within subsegments to enhance separation of segments, such as numbers 0010 through 0990 for subsegment one and numbers 3000 through 3990 for subsegment three.

SPS I-4.4.3.5 Continuation Cards

a) Clear

SPS I-4.4.3.6 Statements

a) Clear

It should not be necessary to start executable code in column 10 if appropriate comment lines are used to offset descriptive material.

SPS I-4.4.3.6.1 Assignment Statements

a) The example does not show indentation by six columns. It would seem that proper interpretation of the guideline would call for the indentation of the continuation lines beyond the equal sign following the variable name.

SPS I-4.4.3.6.2 Common Statements

a) Clear

SPS I-4.4.3.6.3 Type Statements

a) Clear

b) The rule of using one line for each dimensioned variable could be complied with. However, 4 problems have been experienced:

All 4 disadvantages result from the length of program listing generated by this technique.

The first is the difficulty in reviewing the program. It is often desirable to observe the entire printout which becomes excessively long and unmanageable with the dimension information strung out as suggested. Experience has shown that a listing of four pages is the maximum that is easy to handle. Programs of reasonable complexity often will fill all 4 pages with the dimensional information and leave no room for executable code. This runs counter to the basic philosophy of structured programming, namely to bring the program into clear view.

The second and third problems relate to the expense incurred in printing the programs. All 3 environments used by the small contractor: in-house, time sharing and Government computing systems, will increase their supplies cost inversely proportionately to the amount of information contained on each line of listings.

The third problem is not as readily apparent but will show up in increased computing cost for time share services. The commercial time share vendor recoups his costs by charging users proportionately for the amount of resources employed. One of the parameters in the generally complicated cost formula is the number of I/O operations performed. Printed output is usually charged on a per-line basis. Therefore, if three dimensioned variables were allowed for each line, the cost for this portion of the listings would be reduced by 66%. Appropriate spacing between the names would maintain the readability of the listing. Figure II.3.3-2 shows the guideline method and Figure II.3.3-3 contains the alternative suggested.

```
      REAL      AAAAAA(15)
     1         ,BBBBBB(12)
     2         ,CCCCCC(55)
     3         ,DDDDDD(45)
     4         ,EEEDDD(10)
     5         ,FFFFFF(13)
     6         ,SP0001(32)
     7         ,SP0002(35)
     8         ,SP0003(27)
     9         ,SP0004(13)
     A         ,SP0099(19)
     B         ,SP0100(32)
     C         ,TX0001(15)
     D         ,TX0002
     E         ,TX0005
     F         ,TX0010
     G         ,TX0011
     H         ,VERF  (105)
```

Figure II.3.3-2. Structured Programming guideline method of applying TYPE statement.

```
      REAL
     1         AAAAAA(15)      ,BBBBBB(12)      ,CCCCCC(55)
     2,        DDDDDD(45)      ,EEEDDD(10)      ,FFFFFF(13)
     3,        SP0001(32)      ,SP0002(35)      ,SP0003(27)
     4,        SP0099(19)      ,SP0100(32)      ,TX0001(15)
     5,        TX0002          ,TX0005          ,TX0010
     6,        TX0011          ,VERF  (105)
```

Figure II.3.3-3. Suggested alternative method of applying TYPE statement.

The fourth problem also relates to the increased costs of utilizing the proposed type statements. Storage costs for longer programs are naturally higher than for smaller programs. Efficient storage methods are available on commercial time sharing systems as shown in Figure II.3.3-4 where a 13 to one saving was realized. However, this is often not the case on either Government or small in-house computer systems.

```
14.35.35 >L SMPL1 FORTRAN
SMPL1     FORTRAN    P       013

14.36.24 >E SMPL1 FORTRAN OUTBY PACK
EDIT:
>FIL

14.36.55 >L SMPL1 FORT*
FILENAME FILETYPE MODE   ITEMS
SMPL1     FORTRANP  P       001
SMPL1     FORTRAN   P       013

14.37.13 >EDIT SMPL1 FORTRANP INBY UNPACK
EDIT:
>P
C       SAMPLE FORTRAN FILE
SMP00010
>FIL
```

Figure II.3.3-4. PACKING and UNPACKING of program and data files on IBM 370/168 System used by W. W. *Gaertner Research, Inc.* The SMPL1 FORTRAN file is compressed by a factor of 13.

SPS I-4.4.3.6.4 FORMAT Statements

a) Clear

SPS I-4.4.3.6.5 READ or WRITE Statements

a) Clear

SPS I-4.4.3.6.6 IF Statement

a) Clear

Putting the GOTO of an IF statement on a separate line with the same indentation as the multiple conditions hides it from view. Perhaps this is the

desired effect for structured GOTOless programming. However, a programmer reading the listing would want to know three items of information about the statement:

1. The object of the statement. In this case the "IF" portion is clearly visible.

2. The conditions of the statement. Below the "IF" the test parameters are clearly indented.

3. The action the statement will take. The "GOTO", however, is buried with the test parameters.

A technique to expose the third item would be to place the GOTO portion to the right of the last test condition as shown in Figure II.3.3-5. It can be seen from this illustration that the required action of the statement is immediately evident.

```
      IF (A.GT.B .OR.
     1    C.LT.D .AND.
     2    X.EQ.J      ) GOTO 0100
```

Figure II.3.3-5. Suggested technique to expose the action of an IF statement.

SPS I-4.4.3.6.7 DATA Statement

a) Clear

It would not detract from the program listing readability to put multiple DATA Statements on a line.

SPS I-4.5.2.1 IFTHENELSE

a) Clear. There is no standard here. The implementation of IFTHENELSE is left to the programmer.

SPS I-4.5.2.2.1 DOWHILE

a) Clear

SPS I-4.5.2.2.2 DOUNTIL

a) Clear

SPS I-4.5.2.3 CASE Figure

a) Clear

The object of the CASE statement is clearly available in COBOL but under a different name that states exactly what should be accomplished, namely GOTO..... DEPENDING ON .....

SPS I-4.5.2.4 Include

a) Clear

SPS I-4.5.3.1 Restricted ANS COBOL Statement Usage

a) Clear

SPS I-4.5.3.2 Program Organization

a) Clear

SPS I-4.5.3.3 Comments

a) Clear

SPS I-4.5.3.4 Indentation and Formatting Conventions

a) Clear

SPS I-4.6.2 TACPOL

SPS I-4.6.2.1 IFTHENELSE

a) Clear

SPS I-4.6.2.2.1 DOWHILE

a) Clear

SPS I-4.6.2.2.2 DOUNTIL

a) Clear

SPS I-4.6.2.2.3 TACPOL DO (Compound DO)

a) Clear

SPS I-4.6.2.3 Case

a) Clear

SPS I-4.6.2.4 Include

a) Clear

SPS I-4.6.3.1 *Restricted TACPOL Statement Usage*

a) Clear

SPS I-4.6.3.2 Program Organization

a) Clear

SPS I-4.6.3.3 Comments

a) Clear

SPS I-4.6.3.4 Indentation and Formatting Conventions

a) Clear

SPS I-4.7 AN/GYK-12 ASSEMBLY Language (MOL)

SPS I-4.7.1 General Comments

a) Clear

SPS I-4.7.2.1 IFTHENELSE

a) Clear

SPS I-4.7.2.2 DO

a) Clear

SPS I-4.7.2.2.2 DO Indexing

a) Clear

SPS I-4.7.2.3 Case

a) Clear

SPS I-4.7.2.4 Include

a) Clear

SPS I-4.7.3.1 Restricted MOL Statement Usage

a) Clear

SPS I-4.7.3.2 Program Organization

a) Clear

SPS I-4.7.3.3 Comments

a) Clear

SPS I-4.7.3.4 Indentation

a) Clear

SPS I-4.8 IBM S/360 Assembly Language

SPS I-4.8.2.1 IFTHENELSE

a) clear

SPS I-4.8.2.2 DO

a) Clear

SPS I-4.8.2.2.2 Indexing DO

a) Clear

SPS I-4.8.3.3 Case

a) Clear

SPS I-4.8.2.4 Include

a) Clear

SPS I-4.8.3.1 Restricted S/360 Assembly Changing Statement Usage

a) Clear

SPS I-4.8.3.2 Program Organization

a) Clear

SPS I-4.8.3.3 Comments

a) Clear

SPS I-4.8.3.4 Indentation

a) Clear

SPS I - Appendix A Efficiency Considerations

SPS I-A.1 Introduction

a) Clear

SPS I-A.2 Compiler Code Generation Inefficiencies

a) Clear

SPS I-A.3 Simulation Inefficiencies

a) Clear

SPS I-A.4 Improper Source Language Usage

a) Clear

SPS I-A.5 Virtual Storage

a) Clear

SPS I-A.6 Efficiency Summary

a) Clear

The efficiency of structured versus unstructured coding has not been fully verified by actual case studies. They must wait until precompilers are available.

One area where Structured Programming may create problems is that of real-time applications. These are generally written in assembly or machine language because of the inefficiencies introduced by any higher-order language. It is quite likely that the SP coding conventions will introduce certain inefficiencies of execution which may adversely affect the ability to meet response times in real-time applications. Therefore, a study specifically dealing with Structured Programming for real-time applications might be indicated.

SPS I - Appendix B Additional Structuring Capabilities

SPS I-B.1 Introduction

a) Clear

SPS I-B.2 Search

a) Clear

SPS I-B.3 DOEXIT

a) It is not clear from its wording or the example shown in page B-4 what the DOEXIT verb is attempting to accomplish. The indication from the coding is that (q) is tested at the first occurrence of DOEXIT. In reality the condition (p) is tested at this point due to the fact that a DOUNTIL has initiated the segment.

SPS I-B.4 Compound DO Loop

a) Clear

SPS I-B.5 Expanded Case Statement

a) Clear

SPS I-B.6 Interrupt Handling

a) Clear

SPS IV Data Structuring Study

SPS IV - Section 1 Introduction

SPS IV-1.1 Background

Clear

SPS IV-1.2 Report Organization

Clear

SPS IV-1.3 Conclusions

Clear

1. "No single technique or set of techniques has been shown to be either necessary or sufficient".

2. "C.A.R. Hoare and J. Earley works constitute first steps toward a formalized theorem, but much remains to be done".

3. E.W. Dijkstra - attentive to data flow throughout the design process.

4. Use what's available from Hoare, Earley and Dijkstra and Structured Programming.

5. Data definition and/or data flow techniques not in violation or limited by Structured Programming.

SPS IV-1.4 Recommendations

1. Add the INCLUDED BY table

2. Provide Training

3. Develop standards, etc.

SPS IV - Section 2 Literature Survey

SPS IV-2.1 Introduction and Ground Rules

Clear - note ground rules

SPS IV-2.2 E. W. Dijkstra

Clear, basic paper and levels of abstraction and semaphores

SPS IV-2.3 C.A.R. Hoare

Clear - 8 data categories defined

SPS IV-2.4 N. Wirth

Clear - Data is refined, decomposed and structured along with the task

SPS IV-2.5 D. L. Parnos

Clear

"Dr. Parnos points out that the proposed technique is not compellingly better for small projects (those engaging a single team)".

SPS IV-2.6 J. Earley

Clear

"optimize the dynamic usage, rather than the static representation" by utilization of "v-graph".

SPS IV-2.7 H. D. Mills

Clear

"technique for associating data structures with the program segments resulting from the top down process"

SPS IV-2.8 B. H. Liskov

Clear

expansion of "levels of abstraction" and application

SPS IV-2.9 S. E. Madnick and J. W. Alsop II

Clear

application of "levels of abstraction"

SPS IV-2.10 W.P. Stevens, G.J. Myers and L.L. Constantine

Clear

1) minimize the "degree of coupling" of data between modules

2) five steps to implement input, process and output.

SPS IV - Section 3 Problem Analysis

SPS IV-3.1 Introduction

Clear

SPS IV-3.2 Data Handling Problems

SPS IV-3.2.1 Language Incompatibility

Clear

SPS IV-3.2.2 Data Reference Minimization

Clear

SPS IV-3.2.3 Usage Accounting

Clear

SPS IV-3.2.4 Data Retention vs. Regeneration

Clear

SPS IV-3.2.5 Invariance Determination

Clear

SPS IV-3.2.6 Destructive Interaction

Clear

SPS IV-3.2.7 Difficult Development Conditions

Clear

SPS IV - Section 4 Data Techniques

SPS IV-4.1 Introduction

Clear

SPS IV-4.2 Data Declaration Techniques

Clear

A problem that can arise when the data segment is separated from the program segment is the obscurity of the data definitions. For example, when mixed-mode arithmetic is carried out, many compilers will not flag the occurrence with a warning message. The results of the operation may be incorrect when the program executes and may not be evident until later in the processing. The mixed-mode statement would not be readily apparent to the programmer unless the data definitions were placed close to the erroneous code.

SPS IV-4.2.1 Data Declaration Statement

Clear

The data handling and manipulative techniques of a language are important considerations in the selection of the language or languages to be used for a given application. Proper design preparation limits the need for multiple languages. Rather than deal with the inherent design differences of the various languages which the SPS clearly explains, the small contractor could tend to try to exhausting the capabilities of a single selected language before resorting to other languages. It is less likely that the small contractor will apply multiple languages in accomplishing a given task and encounter the problem of multiple data definitions of the same data. However, the subsection does present a viable solution when required.

SPS IV-4.2.2 Data Definitions as Source Segments

This subsection defines "global" and "local" data. It also describes good programming conventions that the small contractor would usually adopt. However, the information in its last paragraph is confusing. In particular, the statement "In a controlled top down programming environment (one in which source segments are scanned for INCLUDE statements) it is necessary to provide an identification mechanism for data segments, so that the programming support library can discriminate data and program segments to insure that the INCLUDE is present before the included segment is entered."

The interpretation of the above sentence is that program segments in the programming support library are not distinctly identified but run continually and it is up to the programmer to select a series of statements from the library. However, data segments must be entered with start and end identifiers so they may be selected by that identifier.

SPS IV-4.3 Data Structures

SPS IV-4.3.1 Data Structure Definition

References to languages not covered by the Series might be eliminated.

SPS IV-4.4 Data Sets

SPS IV-4.4.1 Data Set Usage Accounting

Clear and most likely very useful

SPS IV-4.4.2 Data Set Storage Selection

Clear

SPS IV-4.4.3 Data Set Regeneration

Clear

SPS IV-4.4.4 Data Set Staging

Clear

SPS IV-4.5 Recording Data Usage and Invariance

Clear

SPS IV Appendix A CO-ROUTINE DESIGN

Clear

What does this have to do with Data Structuring?

SPS IV Appendix B Subroutine and Macro Usage Accounting

Clear

SPS IV-B.1 Accounting Technique

Clear

Rather than confuse the issue by utilizing one verb to signify CALL, INCLUDE and MACRO, it would be much clearer to let an accounting routine determine the usage of the variables, which is the desired goal. If it is not desirable to develop such a routine, which would be the case with a small contractor accessing a time sharing system, the EDITOR capability to search an entire file for the occurrence of a string could be used.

SPS IV-B.2 Librarian Macro Language

Clear

II.3.4 Chief Programmer Team

The subject of the Chief Programmer Team is addressed in Volume X, Chief Programmer Team Operations Description of the Structured Programming Series. This Chapter II.3.4 provides detailed comments, examples and recommendations concerning any chapter within the Structured Programming Series where the subject of the Chief Programmer Team is mentioned.

SPS X Chief Programmer Team Operations Description

SPS X Section 1 Introduction

SPS X-1.1 Background

Clear

SPS X-1.2 Report Organization

Clear

SPS X-1.3 Conclusions

Clear

SPS X-1.4 Recommendations

Clear

SPS X Section 2 CPT Description

SPS X-2.1 Introduction

Clear

note - highly competent people

SPS X-2.2 Team Organization

Clear

SPS X-2.3 Core Members

The chief programmer as described in Volume X of the SP Series is a senior programmer with management capabilities. The small contractor often has a core of chief programmers each with a solid software background in various languages from machine code to HOLs. In addition, these individuals usually possess broad knowledge of various operating systems and hardware. They usually possess the management capabilities for working with both backup programmers and consultants. As a matter of fact the majority of programming personnel employed by a small contractor are capable of exercising the duties and responsibilities of the chief programmer, even though they may be the sole program-

mer on a specific project.

The backup programmer at a small contractor's installation is a peer of the chief programmer. As the scope of the projects vary, the chief and backup programmer often switch their tasks depending on which is more competent for the particular project.

The secretary/librarian member of the CPT is usually a secretary who, in general, does not work full time on programming support, since the need for this function is greatly reduced in the small-contractor environment where programming is generally carried out on-line rather than in batch mode. The handling and storage of cards is generally eliminated.

SPS X-2.2.2 Support Members

The CPT of the small contractor utilizes support members on a part time basis more often than on a full time basis.

SPS X-2.3 Support Technology

Top down programming has been used by the competent small contractor for many years.

SPS X-2.4 Variations on Team Structure

Clear

SPS X-2.4.1 Core Members

Clear

SPS X-2.4.2 Support Members

Clear

SPS X-2.4.3 Support Technology

Clear

SPS X-2.4.4 Team Levels

Clear

SPS X-2.5 Managerial Implications

Clear

SPS X Section 3 Team Member Qualifications

SPS X-3.1 Introduction

Clear

SPS X-3.2 Core Members

The small contractor is very likely to employ only highly competent personnel to form a core group to direct and perform the programming projects. These individuals by necessity must also be highly motivated and be self-starters. Lacking these attributes the small contractor is not likely to survive.

SPS X-3.3 Support Members

Clear

SPS X Appendix A Job Description

SPS X-A.0 Introduction

Clear

SPS X-A.1 Core Members

SPS X-A.1.1 Chief Programmer

Clear

SPS X-A.1.2 Backup Programmer

Clear

SPS X-A.1.3 Secretary/Librarian

Clear

SPS X-A.2 Support Members

SPS X-A.2.1 CPT Support Programmer

Clear

SPS X-A.2.2 CPT Support Analyst

Clear

II.3.5 Higher-Order Languages

The subject of Higher-Order Languages (HOL) is addressed in Volumes I, Programming Language Standards, II, Precompiler Specifications; III, Precompiler Program Documentation and XIV, Software Tool Impact of the Structured Programming Series. This Chapter II.3.5 provides detailed comments, examples and recommendations concerning any Chapter within the Structured Programming Series where the subject of Higher-Order Languages is mentioned. An exception is Volume I, Programming Language Standards which was covered in Chapter II.3.3.

SPS II Precompiler Specifications

SPS II-2.1 Precompiler Description

SPS II-2.1.1 Definition

a) Clear

SPS II-2.1.2 Purpose

a) Clear

SPS II-2.1.3 General Description

a) Clear, this is a summary of Vol. I, section 5

SPS II-2.2 Functional Requirements

a) Clear. This reiterates Vol. I, sections 5 and 6.

SPS II-3. Precompiler Input Spec

SPS II-3.1 Introduction

Clear

SPS II-3.2 ANS COBOL

a) Clear (introduction)

SPS II-3.2.1 IFTHENELSE

a) Clear

SPS II-3.2.2 DOWHILE/DOUNTIL

a) Clear

SPS II-3.2.3 Case

a) Clear

SPS II-3.2.4 COBOL Precompiler Restrictions

SPS II-3.2.4.1 Reserved Words

a) Clear

SPS II-3.2.4.2 Paragraph Names

a) Clear

SPS II-3.2.4.3 Structuring Verb Sets

a) Clear

SPS II-3.2.4.4 Other Restrictions

a) Clear

SPS II-3.3 Introduction

a) Clear

SPS II-3.3.1 IFTHENELSE

a) Clear

SPS II-3.3.2 DOWHILE/DOUNTIL

a) Clear

SPS II-3.3.3 Case

a) Clear

SPS II-3.3.4 FORTRAN Restriction

a) Clear

SPS II-3.4 JOVIAL J3

SPS II-3.4.1 IFTHENELSE

a) Clear

SPS II-3.4.2 DOWHILE/DOUNTIL

a) Clear

SPS II-3.4.3 Case

a) Clear

SPS II-3.4.4 JOVIAL J3 Restrictions

a) Clear

SPS II-3.5 JOVIAL J73 - J73/1

SPS II-3.5.1 IFTHENELSE

a) Clear

SPS II-3.5.2 DOWHILE/DOUNTIL

a) Clear

SPS II-3.5.3 Case

a) Clear

SPS II-3.5.4 Restrictions

a) Clear

SPS II-3.6 TACPOL

SPS II-3.6.1 IFTHENELSE

Clear

SPS II-3.6.2 DOWHILE/DOUNTIL

Clear

SPS II-3.6.3 Case

Clear

SPS II-3.6.4 TACPOL Restrictions

Clear

SPS II-3.7 AN/GYK-12 Assembler Language (MOL)

SPS II-3.7.1 IFTHENELSE

Clear

SPS II-3.7.2 DO Figures Macros

Clear

SPS II-3.7.3 Case

Clear

SPS II-3.7.4 Restrictions

Clear

SPS II-3.8 IBM S/360 Assembler Language

SPS II-3.8.1 IFTHENELSE

Clear

SPS II-3.8.2 DO

Clear

SPS II-3.8.3 Case

Clear

SPS II-3.8.4 Restrictions

Clear

SPS II-3.9 INCLUDE Capability

Clear

SPS II-4. Precompiler/Macro Output

SPS II-4.1 Introduction

Clear

SPS II-4.2 COBOL

SPS II-4.2.1 IFTHENELSE

Clear

SPS II-4.2.2 DOWHILE/DOUNTIL

Clear

SPS II-4.2.3 Case

Clear

The placement of the GOTO ... DEPENDING ON I is unfortunate. It is expected that structured code would be sequential as much as possible whereas the precompiler/macro generation of the CASE figure causes a jump forward and then back.

SPS II-4.2.4 NOTE Sentences

Clear

SPS II-4.3 FORTRAN

SPS II-4.3.1 IFTHENELSE

Clear

SPS II-4.3.2 DOWHILE/DOUNTIL

Clear

SPS II-4.3.3 Case

Clear

Same comment as on COBOL Case

SPS II-4.4 JOVIAL J3

SPS II-4.4.1 IFTHENELSE

Clear

SPS II-4.4.2 DOWHILE/DOUNTIL

Clear

SPS II-4.4.3 Case

Clear

SPS II-4.5 JOVIAL J73 - J73/1

SPS II-4.5.1 IFTHENELSE

Clear

SPS II-4.5.2 DOWHILE/DOUNTIL

Example at top of page Vol II, 4-15 is not clear. Where is the DOWHILE - ?

SPS II-4.5.3 Case

Same comment as COBOL case. Also labels are buried in code and must be searched for.

SPS II-4.6 TACPOL

SPS II-4.6.1 IFTHENELSE

Clear

SPS II-4.6.2 DOWHILE/DOUNTIL

Clear

SPS II-4.6.3 Case

Same comment as "COBOL Case"

SPS II-4.7 AN/GYK-12 Assemble Language (MOL)

"Ideal to be achieved"

SPS II-4.7.1 IFTHENELSE

Clear

SPS II-4.7.2 DO

Clear

SPS II-4.7.3 Case

Clear

SPS II-4.8 IBM S/360 Assembly Language

SPS II-4.8.1 IFTHENELSE

Clear

SPS II-4.8.2 DO

Clear

SPS II-4.8.3 Case

Clear

SPS II Section 5 Precompiler Processing

SPS II-5.1.1 Purpose

Clear

SPS II-5.1.2 References

Clear

SPS II-5.2 Summary of Requirements

SPS II-5.2.1 Program Description

Clear

SPS II-5.2.2 Program Function

Clear

SPS II-5.2.2.1 Accuracy and Validity

Clear

SPS II-5.2.2.2 Timing

Clear (there are none)

SPS II-5.2.3 Flexibility

Clear

SPS II-5.3 Environment

SPS II-5.3.1 Support Software

Clear

SPS II-5.3.2 Interfaces

Clear

SPS II-5.3.3 Storage

Clear (not defined at this time)

SPS II-5.3.4 Security

Clear

SPS II-5.3.5 Controls

Clear

SPS II-5.4 Design Data

SPS II-5.4.1 General Operating Procedures

Clear

SPS II-5.4.2 Inputs

Clear

The "option card" should not be necessary in a time-sharing environment. It may be listed on the command line.

SPS II-5.4.3 Outputs

Clear

SPS II-5.4.3.1 Error Messages

Clear

SPS II-5.4.3.2 COBOL Precompiler Error Messages

Clear

SPS II-5.4.3.2.1 Errors which DO NOT terminate processing

Clear

SPS II-5.4.3.2.2 Errors Which Terminate Processing

Clear

SPS II-5.4.4 Data Environment

Clear

SPS II-5.4.5 Critical Algorithms

Clear, intro to INCLUDE Capability

SPS II-5.4.5.1 Generalized INCLUDE Capability

Clear

SPS II-5.4.5.2 The Include Algorithm

Clear

SPS II-5.4.5.3 Exception Processing

Clear

In a time-sharing environment that allows execution of prestored code such as 'EXEC' files, a precompiler would be relatively easy to implement. A stored control file, as shown in Figure II.3.5-1, would accept the structured FORTRAN source file name, precompile it and if no fatal errors resulted, compile the preprocessed source file.

The example could easily be modified to execute any language precompiler, for instance, the string of characters 'FORTRAN' could be changed to 'COBOL' and line 9 changed to PRECOMCO for the file to precompile a structured COBOL program.

```
11.54.32>PRINT PRECOMFO EXEC

&COMMENT PREFORT
&COMMENT PRECOMPILE AND COMPILE A STRUCTURED FORTRAN SOURCE
&COMMENT
&COMMENT ARGUMENTS: THE STRUCTURED FORTRAN FILENAME
&COMMENT THE FILETYPE IS ASSUMED SFORTRAN
&COMMENT
FILEDEF 1 DSK &1 SFORTRAN
FILEDEF 2 DSK &1 FORTRAN
PRECOMFO
&IF &INDEX0 EQ 0 &GOTO -COMPILE
&PRINT FATAL ERROR IN PRECOMPILE
&PRINT COMPILE NOT DONE
&EXIT
&COMMENT
-COMPILE
FORTRAN &1
&EXIT

11.55.22>
```

Figure II.3.5-1. Example of simple stored code to precompile and (if no fatal precompiler errors occur) compile the structured FORTRAN source.

More elaborate modifications to the file would allow the inclusion of all information suggested for the "options card", Volume II subsection 5.4.2, to be included in the command line to the file. Another possibility would be to have one command to automatically precompile a structured source file in any higher level language. The versatility of the EXEC facility would allow such an approach.

SPS II-5.4.6 HIPO Diagrams

Clear (gives flow chart of precompiler)

SPS III ANS COBOL Precompiler Documentation

SPS III-1.1 Background

Clear, reference Vol 2, para. 4.2

SPS III-1.2 Report Organization

Clear

SPS III Section 2 Precompiler Description

SPS III-2.1 Precompiler Objectives

Clear

SPS III-2.2 Precompiler Inputs

Clear a) option card - b) source code

SPS III-2.3 Precompiler Outputs

Clear

SPS III Section 3 COBOL Precompiler Input

SPS III-3.1 General Information

Clear

SPS III-3.2 Precompiler Input Formats

SPS III-3.2.1 Case Figure

Clear

SPS III-3.2.2 DOWHILE/DOUNTIL Figures

Clear

SPS III-3.2.3 IFTHENELSE Figure

Clear

SPS III-3.2.4 OPTION Card

Clear

SPS III-3.2.5 Unstructured Code .ON/OFF. Indicators

Clear

SPS III Section 4, Precompiler OUTPUT

SPS III-4.1 General Information

Clear

SPS III-4.2 Precompiler Code Generated

SPS III-4.2.1 Case Figure

Clear

SPS III-4.2.2 DOWHILE/DOUNTIL Figures

Clear

SPS III-4.2.3 IFTHENELSE Figure

Clear

SPS III-4.2.4 Note Sentences

Clear

SPS III Section 5, COBOL Compiler Restrictions

SPS III-5.1 Reserved Words

Clear

SPS III-5.2 Paragraph Names

Clear

SPS III-5.3 Structuring Verb Sets

Clear

SPS III-5.4 Other Restrictions

Clear

SPS III Section 6, Error Messages

SPS III-6.1 General Information

Clear

SPS III-6.2 Messages

SPS III-6.2.1 Errors Which Do Not Terminate Processing

Clear

SPS III-6.2.2 Errors Which Terminate Processing

Clear

SPS III Section 7, Precompiler Dependencies

SPS III-7.1 Introduction

Clear

SPS III-7.2 Potential Implementation Dependencies

Clear

Just as each manufacturer has developed his own compiler or compilers, it is expected that precompilers will have to be tailored to the machines on which they are destined to run. It is recommended as a minimum to make 3 versions of precompilers available, one for each of the following manufacturer's machines: IBM, CDC, DEC.

SPS III Appendix A S/360 Precompiler Installation

SPS III-A.1 S/360 Installation Procedure

Clear

SPS III Appendix B Source Code Data Storage and Execution Seq.

SPS III-B.1 Introduction

Clear

SPS III-B.2 Data Storage Push Down Stacks

Clear

SPS III-B.3 Execution Sequence of Source Segments

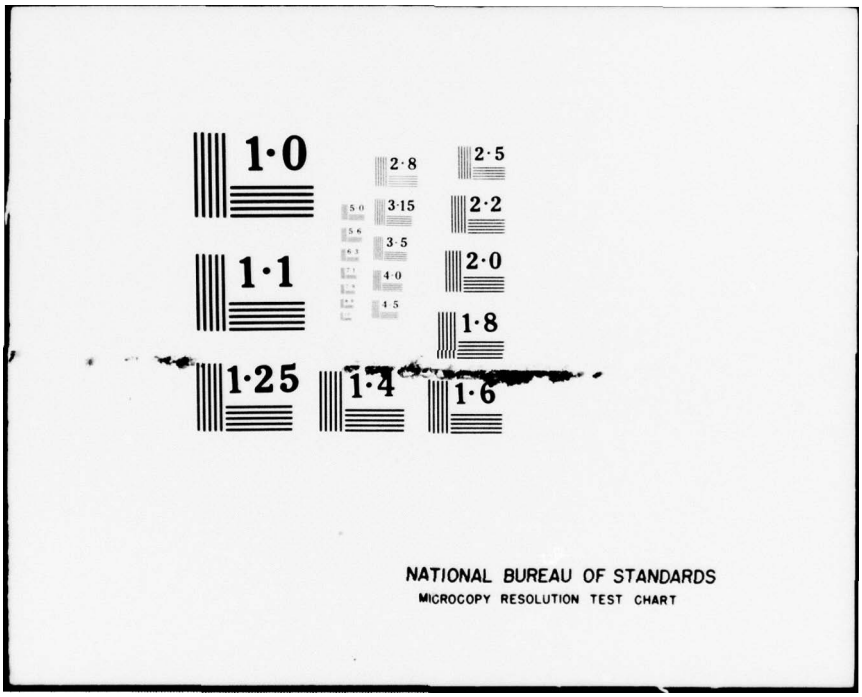
Clear

AD-A040 828 GAERTNER (W W) RESEARCH INC STAMFORD CONN F/G 5/1
IMPACT OF STRUCTURED PROGRAMMING STANDARDS ON SMALL GOVERNMENT --ETC(U)
MAY 77 W W GAERTNER F30602-76-C-0390
UNCLASSIFIED RADC-TR-77-162-VOL-2 NL

3 OF 3
ADA
040828

END

DATE
FILMED
7-77


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6
6·3
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

SPS XIV Software Tool Impact

SPS XIV Section 1 Introduction

SPS XIV-1.1 Background

Clear

SPS XIV-1.2 Report Organization

Clear

SPS XIV-1.3 Conclusions

Clear

SPS XIV-1.4 Recommendations

Clear

SPS XIV Section 2 Precompilers

SPS XIV-2.1 Definition

Clear

SPS XIV-2.2 SP Precompiler

Clear

SPS XIV-2.3 Advantages/Disadvantages

An advantage of a precompiler is that it is an "expeditious solution" that requires "a relatively small investment and is implementable in a short time span". If the figures presented in Volume XIII are used as a basis of comparison, then the small contractor can expect to use 750 man hours and a total cost of $20,000 per language for a precompiler. It is not logical for the Government to support many such precompiler efforts, one for each small contractor. It is recommended that the small contractor be supplied with a precompiler in the target language that will compile in the standard compiler for the language. Assuming that the precompiler is properly documented, the small contractor should not have to expend more than one man week to make it operational on his in-house system.

SPS XIV Section 3 Precompiler Impact During Program Development

SPS XIV-3.1 Other Preprocessor Interfaces

Clear

SPS XIV-3.2 Compiler Output Listings

Clear

SPS XIV-3.3 On-Line Programming

Clear

SPS XIV-3.4 Path Analysis

Clear

SPS XIV-3.5 Execution Statistical Analysis Program

Clear

SPS XIV-3.6 Statement Identification Considerations

Clear

SPS XIV-3.7 Tracer Tools

Clear

SPS XIV-3.8 Flow Charting Tools

Clear

SPS XIV-3.9 Cross-Reference Tools

Clear

SPS XIV Section 4 Precompiler Impact on the Operational Environment

SPS XIV-4.1 PSL

Another point that should be emphasized is that the two source files, precompiler input and output need not be stored on-line but should be retained off-line, preferably on magnetic tape during maintenance of the system.

SPS XIV-4.2 Optimizing Compiler

Clear

SPS XIV-4.3 Source Code Optimizer

Clear

SPS XIV-4.4 Conversion Aids

This paragraph assumes that there is one and only one precompiler for a given HOL. It seems more likely that different precompilers will exist for hardware from different manufacturers. Volume XIII mentioned that the COBOL precompiler written for an IBM 360 was converted to the HIS computer with relative ease.

SPS XIV Section 5 Future Software Tools

SPS XIV-5.1 Introduction

Clear

SPS XIV-5.2 Indentation Aids

Clear

SPS XIV-5.3 Future Compiler Development Impacts

Clear

II.3.6 Subcontracting Provisions Imposed on the Small Contractor

The subject of Subcontracting Provisions Imposed on the Small Contractor is not addressed in any Volume of the Structured Programming Series. This Chapter II.3.6 provides some comments concerning any aspect of the Structured Programming Series which would relate to Subcontracting Provisions Imposed on the Small Contractor.

Subcontracting can occur under a wide range of circumstances: If the Small Contractor simply supplies programming personnel using the large company's computer facilities, all the SP tools and procedures would be available, and the Small Contractor's personnel would only have to be familiar with the TDSP Standards (which presumably would be the case already if the company is involved in Government software development work). If the Small Contractor does not use the large company's facilities, and if the subcontracted software development is a stand-alone item which need not be integrated with a large overall software package, the Small Contractor would use the approaches normally applied to any Government prime contract, namely the use of in-house minicomputers or large-machine time-share services. In this case the Small Contractor would use a standard TDSP procedures and tools, just like on a Government prime contract. The most delicate situation would arise if the software to be developed by the Small Contractor (without the use of the large company's machine) must ultimately be integrated into a single large software package developed by the large company. Standard procedures would, of course, be used to assure compatibility of the subcontracted software with the main software package, such as defining all the formats for data files which are jointly used by the various program sections, and possibly a coordination of nomenclature such as variable names, etc. Additional problems will arise when the large company wants to integrate the subcontracted programs not only into the deliverable software, but also into their in-house SP system. In this case the subcontracted software would have to be compatible with the large company's PSL system, and the small company may have to deliver, in addition to the working software itself, substantial information of statistical, historical and management-information nature. The degree to which this is possible, desirable or necessary will have to be worked out as part of the subcontracting negotiations. In some cases the large company may supply copies of their standard programming aids.

II.3.7 Software Development Tools

The subject of Software Development Tools is addressed in Volumes I, Programming Language Standards; VIII, Program Design Study and XII, Training Materials of the Structured Programming Series. This Chapter II.3.7 provides detailed comments, examples and recommendations concerning any chapter within the Structured Programming Series where the subject of Software Development Tools are mentioned.

SPS I-2.2 Top Down Structured Programming

a) Clear

Use structured logic sequences as stubs. Create a plan (Top Down Programming). Utilize "Tree Structure" for plan. First define (plan) main routine that has modes (stubs), second plan stubs. Define Data Base. Define input and output data structures for interfaces.

b) The use of good planning techniques is fundamental to the success of any small contractor.

c) There should be no problem complying with a standard for a plan which is already a way of life with the small contractor.

d) Recommendations would be highly dependent upon the scope of the contract. They might include such items as

1) hardware restrictions
2) desired output formats (if program preprocesses data for an existing system)

e) An example of how the small contractor uses TDSP for software contracts is described in the following: The purpose of the particular contract was to advance the state-of-the-art for electronic military equipment reliability calculations. When the contract began, there was no doubt that the calculations could be automated by programming the equations and providing the necessary tables to the program. It was realized that two types of parameters were needed in the equations, namely, device-dependent values and values which characterize the operating environment. For example, a device-dependent value is the number of gates in an IC, while an operational value is the operating temperature. Both types of information are

required to calculate parts reliability. Collection of the device-dependent data had always been a major effort since there are thousands of military parts whose properties have to be looked up in manufacturers' catalogs. Since this parts information is independent of equipment design and application and could therefore be collected into a large data base which would eliminate the need to look up parts characteristics in data sheets and catalogs. When the project was started MIL-HDBK-217A was the "bible" for military failure-rate calculations. However, it was undergoing a major revision which eventually resulted in MIL-HDBK-217B. Through careful planning the format of the data base was made compatible with the B version and did not require redesign when MIL-HDBK-217B became the official standard.

Subsequently, the reliability-prediction program was designed such that the analyst only had to enter a parts list consisting of part numbers and operational data (but without device characteristics), and the program itself would search the data base for the parts characteristics needed in the actual reliability calculations.

It was decided that there would be 5 major steps to the program:

1) sort
2) binary search
3) reliability calculation
4) summary
5) trade-off

The sort was required so that the user's parts list would be in the same sequence as the data base thus allowing a much more efficient binary search as opposed to a search from a random parts list.

The reliability calculation was designed so that there would be a main driver routine that would call the appropriate subroutine for a particular part. Each part subroutine would be a stub until it was coded to contain the reliability calculation. The summary step was provided to add the failure rates of the individual parts and provide the total failure rate for the module characterized by the input parts list.

The trade-off step allows the reliability engineer to determine the effects of varying the operating environment of the equipment or of applying different quality-assurance procedures to the parts procurement.

The main program of each step was coded and was immediately made operational. The reliability-calculation subroutines were added to step 3 as soon as they were coded and an appropriate data base collected for them. The system remained operational as subroutines were added. The final reliability-calculation package of programs resulted in 12,000 lines of code and a data base of over 2 million characters. Extension of the original program plan resulted in the creation of two more steps, one estimating parts cost, and the second estimating the cost of the packaged equipment. The same top-down approach was employed in the reliability calculation was applied to create the final two steps. The three main programs - reliability calculation, part cost and packaged cost - are quite similar due to the top-down programming plan used. The total package is composed of approximately 20,000 lines of source code.

While the entire development effort took over two years, the top-down approach allowed W. W. Gaertner Research, Inc. to make portions of the program (for the most frequently used parts types such as integrated circuits) operational after a few months of initial coding, and keep the program available to users during the remainder of the development cycle, as its capabilities to analyze an increasing variety of parts types was added. This had the additional advantage that the program was immediately subjected to extensive operational testing.

SPS I-5.0 General Comments

a) Clear

When Structured Programming is fully implemented it must be assumed that Government computer installations which the small contractor may be required to use will have fully operational precompilers.

Commercial time sharing vendors are usually prompt to respond to new trends in programming aids such as precompilers. It would be expected that they would offer this service to their customers in a timely manner.

The in-house computers used by small contractors may pose a problem to the programmer. Manufacturer support would most likely be slower and it would be up to the individual to develop initial pre-compilers.

It is suggested that the Government supply a pre-compiler to small contractors for their in-house computers. It should be written in a higher level language such as FORTRAN. FORTRAN is recommended because it is the most widely supported higher-level language for minicomputers. Also, machine independence can be achieved with a higher-level language as opposed to assembly-level language. Different versions of the precompiler could be made available for the most popular minicomputers so that the user would be able to compile the source and utilize it immediately. The precompiler source would have to be compiled on the standard ANS compiler of the mini and should not rely on any support programs which are not supplied by the manufacturer. However, being written in a structured manner it would also provide a clear example of how to produce a structured program.

SPS I-5.2 Precompiler Input

a) Clear

SPS I-5.3 Precompiler Features

SPS I-5.3.1 IFTHENELSE

a) Clear

SPS I-5.3.2 DOWHILE

a) Clear

SPS I-5.3.3 DOUNTIL

a) Clear

SPS I-5.3.4 Case

a) Clear

SPS I-5.3.5 Include

a) Clear - says look at Vol. I, sec. 6.

SPS I-5.3.6 Indentation

a) Clear

SPS I-5.4 Precompiler Output

SPS I-5.4.1 Compiler Input

a) Clear

SPS I-5.4.2 Simulated Structured Code

a) Clear

SPS I-5.5 Language Considerations

These repeat all that was said in Section 4, Vol. I.

SPS I-5.5.1 JOVIAL J73 and J73/1

a) Clear

SPS I-5.5.2 JOVIAL J3

a) Clear

SPS I-5.5.3 ANS FORTRAN

a) Clear

SPS I-5.5.4 ANS COBOL

a) Clear

SPS I-5.5.5 TACPOL

a) Clear

An added feature of a precompiler would be a PAGE command. This would allow the programmer to artificially insert a TOP-OF-FORM into his listing preceding each segment thus having each new segment start a new page, rather than beginning in the middle of a page.

SPS I-6.0 INCLUDE Capability and Segment Size

SPS I-6.1 Introduction

a) Clear

SPS I-6.2 Software Implementation of INCLUDE

SPS I-6.2.1 Language Processor Include

a) Clear

JOVIAL J73 has ! COPY to do it

SPS I-6.2.2 PSL INCLUDE

a) Clear

SPS I-6.2.3 Precompiler INCLUDE

a) Clear

SPS I-6.3 Segment Size

a) Clear

SPS VIII Program Design Study

SPS VIII Section 1 Introduction

SPS VIII-1.1 Background

Clear

SPS VIII-1.2 Report Organization

Clear

SPS VIII-1.3 Conclusions

Clear

SPS VIII-1.4 Recommendations

Clear

SPS VIII Section 2 Structured Programming Technology Fundamentals

SPS VIII-2.1 Introduction

Clear

SPS VIII-2.2 TDSP

Clear

SPS VIII-2.2.1 SP

Clear

SPS VIII-2.2.2 Use of Subroutines

Clear

SPS VIII-2.2.3 Top Down Programming

Clear

SPS VIII-2.3 PSL

Clear

SPS VIII-2.4 Definitions

Clear

SPS VIII Section 3 Design and Survey

SPS VIII-3.1 Introduction

Clear

SPS VIII-3.2 Flow Charts

Clear

SPS VIII-3.2.1 Definition

Clear

SPS VIII-3.2.2 Description

SPS VIII-3.2.2.1 Purpose

Clear

SPS VIII-3.2.2.2 Flow Chart Types

Clear

SPS VIII-3.2.3 Advantages/Disadvantages

Clear

Automatic flow charting programs are not mentioned in this section.

SPS VIII-3.3 Decision Tables

SPS VIII-3.3.1 Definition

Clear

SPS VIII-3.3.2 Description

Clear

SPS VIII-3.3.3 Advantages/Disadvantages

Clear

SPS VIII Section 4 PDL

SPS VIII-4.1 Introduction

Clear

SPS VIII-4.2 Definition

Clear

SPS VIII-4.3 Purpose

Clear

SPS VIII-4.4 Advantages

Clear

SPS VIII-4.4.1 Preparation

Clear

SPS VIII-4.4.2 Usability

Clear

SPS VIII-4.4.3 Promotion of TDSP

Clear

SPS VIII-4.5 Disadvantages

Clear

SPS VIII-4.6 PDL Forms

Clear

SPS VIII-4.6.1 Freeform PDL

SPS VIII-4.6.1.1 Description

Clear

SPS VIII-4.6.1.2 Consideration

Clear

SPS VIII-4.6.2 Formal PDL

SPS VIII-4.6.2.1 Description

Clear

SPS VIII-4.6.2.2 Considerations

Clear

SPS VIII-4.6.3 Summary

Application of two PDLs is suggested for a project, a free-format PDL for designers and formal PDL for documentors. If, in fact, the documentors are employed at the end of a project to produce the deliverable PDL code then no advantage over the production of flow charts at the end of the project is gained.

SPS VIII Section 5 PDL Examples

SPS VIII-5.1 Introduction

Clear

SPS VIII-5.2 Pseudo Code

SPS VIII-5.2.1 Description

Clear

SPS VIII-5.2.2 Advantages

Clear

SPS VIII-5.2.3 Disadvantages

Clear

SPS VIII-5.3 Playscript

SPS VIII-5.3.1 Description

Clear

SPS VIII-5.3.2 Advantages

Clear

SPS VIII-5.3.3 Disadvantages

Clear

SPS VIII-5.4 PIDGIN

SPS VIII-5.4.1 Description

Clear

SPS VIII-5.4.2 Advantages

Clear

SPS VIII-5.4.3 Disadvantages

Clear

SPS VIII-5.5 Summary

Clear

SPS VIII Section 6 Other Design Aids

SPS VIII-6.1 Introduction

Clear

SPS VIII-6.2 HIPO

Clear

SPS VIII-6.2.1 Description

Clear

SPS VIII-6.2.1.1 Visual Table of Contents (VTOC)

Clear

SPS VIII-6.2.1.2 HIPO Diagrams

Clear

SPS VIII-6.2.2 Objective/Purpose

Clear

SPS VIII-6.2.3 Considerations

Clear

SPS VIII-6.2.4 Experience

Clear

SPS VIII-6.2.4.1 Functional Requirements

Clear

SPS VIII-6.2.4.2 Program Specifications

Clear

SPS VIII-6.2.5 HIPO Summary

Clear

SPS VIII-6.3 Structured Design

SPS VIII-6.3.1 Objectives/Purpose

Clear

SPS VIII-6.3.2 Description

SPS VIII-6.3.2.1 Relationship Between Modules

Clear

SPS VIII-6.3.2.2 Substance of Module

Clear

SPS VIII-6.3.2.3 Organization of Modules and Decisions

Clear

SPS VIII-6.3.3 Considerations

Clear

SPS VIII-6.3.4 Structured Design Summary

Clear

SPS VIII-6.4 Program Review

SPS VIII-6.4.1 Introduction

Clear

SPS VIII-6.4.2.1 Purposes

Clear

SPS VIII-6.4.3 Types of Program Review Activities

Clear

SPS VIII-6.4.4 Relationship of Program Review to the Program Design Process

Clear

SPS VIII-6.5 Summary

Clear

SPS VIII Section 7 Recommendations for 3 Specific Tools/Techniques

SPS VIII-7.1 Introduction

Clear

SPS VIII-7.2 Recommended PDL

SPS VIII-7.2.1 Description

SPS VIII-7.2.1.1 Sequence

Clear

SPS VIII-7.2.1.2 IFTHENELSE Figure

Clear

SPS VIII-7.2.1.3 DOWHILE Figure

Clear

SPS VIII-7.2.1.4 DOUNTIL Figure

Clear

SPS VIII-7.2.1.5 CASE Figure

Clear

SPS VIII-7.2.1.6 Library Segmentation

Clear

SPS VIII-7.2.2 Justification

Clear

The fact that precompilers do not exist for all HOLs does not seem to bother the authors. Without a precompiler the specified PDL would not have all of the advantages suggested. One point touched upon very briefly but not really highlighted is that HIPO diagrams are as difficult to update and maintain as flow charts.

SPS XII Training Materials

SPS XII Section 1

SPS XII-1. Background

Clear

SPS XII-1.1 Training Material Organization

Clear

SPS XII Section 2

SPS XII-2. Introduction

Clear - Printed copies of the viewgraphs should be included in this Volume.

SPS XII-2.1 Executive Overview

Clear

SPS XII-2.2 Contract Summary

Clear

SPS XII-2.3 Language Standards

Clear

SPS XII-2.4 Precompiler

Clear

SPS XII-2.5 Data Structuring

Clear

SPS XII-2.6 Programming Support Library

Clear

SPS VII-2.7 Program Design Study

Clear

SPS VIII-2.8 Documentation Standards

Clear

SPS XII-2.9 Data Collection and Reporting

Clear

SPS XII-2.10 Estimating

Clear

SPS XII-2.11 Validation and Verification

Clear

SPS XII-2.12 Chief Programmer Team Operation

Clear

II.4 Detailed Comments on Automated Management Tools

II.4.1 Management Reporting Tools

The subject of the Management Reporting Tools is addressed in Volumes IX, Management Data Collection and Reporting and XI, Estimating Software Project Resource Requirements of the Structured Programming Series. This Chapter II.4.1 provides detailed comments, examples and recommendations concerning any chapter within the Structured Programming Series where the subject of the Management Reporting Tools are mentioned.

SPS IX Management Data Collection and Reporting

SPS IX Section 1 Introduction

SPS IX-1.1 Background

Clear

SPS IX-1.2 Report Organization

Clear

SPS IX-1.3 Conclusions

Clear

SPS IX-1.4 Recommendations

Clear

SPS IX Section 2 Management Considerations

SPS IX-2.1 Introduction

Clear

SPS IX-2.2 Management Functions

Clear

SPS IX-2.3 Estimating/Measuring Concerns

Clear

SPS IX-2.4 Impact of SP Technology

Clear

SPS IX-2.5 Summary

Clear

SPS IX Section 3 Functional Description

SPS IX-3.1 Introduction

Clear

SPS IX-3.2 Software Development Cycle

Clear

SPS IX-3.2.1 System Definition

Clear

SPS IX-3.2.2 System Design

Clear

SPS IX-3.2.3 System Implementation

Clear

SPS IX-3.2.4 System Evaluation

Clear

SPS IX-3.2.5 Relationship of Phases

Clear

SPS IX-3.3 Major Functional Requirements

Clear

SPS IX-3.4 Collection and Reporting Levels

Clear

SPS IX-3.5 Data Classes and Types

SPS IX-3.5.1 Background

Clear

Figure 3.5.1 of Volume IX is discussed in this paragraph. In reviewing the figure it is noted that neither resource cost

nor computer utilization are automatically collected. Either of the two or both definitely appear suited for automatic collection. Further discussion will follow in the appropriate subsection reviews.

SPS IX-3.5.2 *Project Environment Definition*

Clear

SPS IX-3.5.3 Module Description Definition

Clear

An example for this type of data is lines of source code for a program. This information can be readily collected on the commercial time sharing system via the LISTF command as shown in Figure II.4.1-1. The field labeled ITEMS is the number of lines in the files listed. The figure shows the results of the command directed to a terminal. However, an option could have been used to create a file containing the information which could later be accessed by a management information reporting system. The command could also be included in an EXEC file that would automatically create the file and run the program as a step in a management report generator. The lines of code in a program is not readily available automatically either in-house or on the Government computer. It is recommended that a lines-of-source-code utility program be developed for the small contractor's in-house computer system.

```
15.41.02>LISTF * DATA
FILENAME  FILETYPE  MODE  ITEMS
B55PRTA   DATA        F      30
B55PRTC   DATA        F      10
B55PRTD   DATA        F      15
B55PRTF   DATA        F      24
B55MASTR  DATA        F      79

15.44.35>
```

Figure II.4.1-1. Display of number of lines of source code on a typical commercial time sharing system.

SPS IX-3.5.4 Computer Utilization Definition

Computer Utilization is defined as computer turnaround time which, as the SP series notes, averages 4 hours on a batch system. The small contractor accessing a time sharing system experiences turnaround times measured in minutes and seconds, not hours. With such short turnaround times it may not be very useful to collect a detailed statistical history.

3.5.5 Resource Cost Definition

Resource cost such as computer time can be and usually is automatically collected by most time sharing systems. It is used in charging for the equipment and resources utilized during a billing period.

SPS IX-3.5.6 Program Production Definition

Clear

SPS IX-3.5.6.1 Reasons for Source Code Updates

Clear

SPS IX-3.5.6.1.1 Program Improvements

The author(s) of this volume do not seem to be fully aware of the contents of previous volumes of the SP Series. They refer to a "test driver" whereas TDSP does not employ test drivers but stubs.

SPS IX-3.5.6.1.2 Functional Requirements changes/Extensions

Clear

SPS IX-3.5.6.1.3 Program Errors

Clear

SPS IX-3.6 Report Classes

SPS IX-3.6.1 Background

Clear

SPS IX-3.6.2 Program Module Statistics

The example of program module statistics reports shows a quantitative summation of the work performed. Analysis of a Figure 3.6-2 indicates that there might be a problem with

the report generator: The lines of source code are listed as 8500 while the lines of COBOL code are 7018 and the lines of assembly code are 1502. The sum of the latter two, it is assumed, should be equal to the first but an error of 20 lines of code exists.

SPS IX-3.6.3 Program Production Statistics

Clear

SPS IX-3.6.4 Computer Utilization

There are only two items contained in this type of report, the number of runs and the turnaround time. The programmer controls the number of runs but has no control over the turnaround time. The small contractor does not experience the long turnaround times indicated in the sample report when accessing on an on_line system. It is recommended that the report for batch-oriented users be further broken down so that turnaround time is shown as two items, namely the time in the queue and the execution time. With this information the bottlenecks in a batch system would be more visable.

SPS IX-3.6.5 Program Structure

Clear

SPS IX-3.6.6 Historical

Clear

SPS IX-3.6.7 Combination

Clear

SPS IX Section 4 Functional Requirements

SPS IX-4.1 Introduction

Clear

SPS IX-4.2 Statement of Functional Requirements

SPS IX-4.2.a Collecting

Counting of input source code is available to the small contractor on the commercial time sharing system via the LISTF command. If either the in house or Government computer are accessed, storage utilized is available but the number of lines

of source code is not. A simple routine could be written to count the lines of source code on the in-house system or the Government system. Storing or modifying existing Actual Data in a MSDB (management statistical data base) is not readily available to the small contractor but if a "milestone-achieved" technique is being employed as an indicator of program completion, tnen the number of lines of code would not be of paramount interest.

The source unit start date and similar information can be gathered on all three systems. The information when gathered could be stored in a MSDB if a routine were available to do so.

Counting the number of compilations cannot be readily accomplished on a time sharing system unless the operating system has a count feature included which none of the three example systems do. It was suggested in Volumes V and VI that the PSL interface to the system be written such that it could perform the appropriate counting. Either method, system counting or PSL interface, could be implemented, but the value of the information should be reconsidered first: For instance, most HOL compilers and even assemblers have extensive error detection capabilities. With the cost of computing dropping and the cost of programmers rising, it often is more cost-effective to let the compiler perform syntax checking than to have a programmer proof read a listing. In this case a large number of compiles would be an indication of cost-effective programming and not of an excessive number of errors.

SPS IX-4.2.b Updating

The functions of updating a MSDB can readily be performed via the editor of the three example systems. Figures throughout this report show the many capabilities of these editors.

SPS IX-4.2.c Accumulating

An accumulation facility for the MSDB is a recommended feature for the small contractor in order to implement the MSDB requirement of TDSP.

SPS IX-4.2.d Reporting

Reporting of MSD is accomplished by retrieving data from the MSDB, temporary Report Data Base and the Archival Project Data Base and presenting it in a specified format. Assuming that the small contractor has collected the data he should

have no problem producing the reports provided appropriate report routines are available. The support software is not presently available. It is recommended that appropriate reporting routines be developed for the small contractor's in-house system. The commercial time sharing system would probably make the appropriate routines available to all users as soon as the demand is sufficient to repay the cost of implementation. The Government computer would most likely have the software available for the small contractor to fulfill Government SP requirements.

SPS IX-4.2.e Archiving

The capability to archive files has been shown previously. The small contractor is able to retain copies of on-line files on off-line storage media, such as magnetic tape and to retrieve the data from the tape.

SPS IX-4.3 Graphic Representation of Functional Requirements

Clear

SPS IX Section 5 Implementation Considerations

SPS IX-5.1 Implementation Recommendation

Clear

For optimum implementation of the MSDB with reporting, the PSL must be implemented. Recommend FORTRAN for small contractor, not RPG.

SPS IX-5.2 Programming Support Library Description

SPS IX-5.2.1 Definition

Clear

SPS IX-5.2.2 Objective

Clear

SPS IX-5.2.3 Purpose

Clear

SPS IX-5.3 User Procedures

Clear

SPS IX-5.4 Cost Factors

SPS IX-5.4.1 Development Costs

It is recommended that the Government supply a standard software package to the small contractor which is written in FORTRAN. This would significantly reduce the implementation cost to the small contractor. The only cost would be installation, which should not be large since a HOL is utilized. FORTRAN is recommended because it is the language most widely supported by the minicomputer manufacturers. Recently COBOL is becoming supported on minicomputers but it is not available for all operating systems. Another advantage of employing FORTRAN as the development language is that it requires a smaller minicomputer configuration than does COBOL. The large computing systems of the commercial time sharing companies and the Government computer support both COBOL and FORTRAN and either could be utilized as an implementation language.

The development of external procedures by the small contractor would be aided by a set of guidelines and specific examples.

SPS IX-5.4.2 Operating Costs

Clear

SPS IX Appendix A Data Item Definitions

SPS IX-A.1. Introduction

Clear

SPS IX-A.2. Project Environment Data Items

Clear

SPS IX-A.3. Module Description Data Items

Clear

SPS IX-A.4. Resource Cost Data Items

Clear

SPS IX-A.5. Computer Utilization Data Items

Clear

The average and actual computer turnaround times would not be relevent report items for the small contractor accessing a time sharing system.

SPS IX-A.6. Program Production Data Items

Clear

SPS IX Appendix B Report Descriptions

SPS IX-B.1.0 Introduction

Clear

SPS IX-B.1.1 Program Module Statistics Class

Clear

SPS IX-B.1.2 Program Production Statistics Class

Clear

SPS IX-B.1.3 Computer Utilization Class

Clear

SPS IX-B.1.4 Program Structure Class

Clear

SPS IX-B.1.5 Historical Class

Clear

SPS IX-B.1.6 Combination Class

Clear

SPS IX-B.2.0 Program Module Statistics Class

Clear

SPS IX-B.3.0 Program Production Statistics Class

Clear

SPS IX-B.4.0 Computer Utilization Class

Clear

SPS IX-B.5.0 Program Structure Class

Clear

SPS IX-B.6.0 Historical Class

Clear

SPS IX-B.7.0 Combination Class

Clear

SPS IX Appendix C Cross Reference Listing

Clear

SPS XI Estimating Software Project Resource Requirements

SPS XI Section 1 Introduction

SPS XI-1.1 Background

Clear

SPS XI-1.2 Report Organization

Clear

SPS XI-1.3 Conclusions of Literature Survey

Clear

SPS XI-1.4 Conclusions - SP Technology Impact on Estimating Methodology

Clear

SPS XI-1.5 Recommendations

Clear

SPS XI Section 2 Literature Survey

SPS XI-2.1 Introduction

Clear

SPS XI-2.2 Estimating - A Management Tool

SPS XI-2.2.1 Definition

Clear

SPS XI-2.2.2 Utility

Clear

SPS XI-2.2.3 Components

Clear

SPS XI-2.2.4 Estimating Problems

Clear

SPS XI-2.3 Types of Estimates

Clear

SPS XI-2.4 Factors Influencing Estimates

SPS XI-2.4.1 Background

Clear

SPS XI-2.4.2 System Development Corporation Studies

Clear

SPS XI-2.4.3 Planning Research Corporation Studies

Clear

SPS XI-2.4.4 Other Studies

Clear

SPS XI-2.5 Estimating Techniques/Guidelines

SPS XI-2.5.1 Background

Clear

SPS XI-2.5.2 Similar Experience Technique

Clear

SPS XI-2.5.3 Quantitative Technique

Clear

SPS XI-2.5.4 Statistical Technique

Clear

SPS XI-2.5.5 Constraint Technique

Clear

SPS XI-2.5.6 Unit of Work Technique

Clear

SPS XI-2.5.7 Computer Resource Estimating Guidelines

Clear

SPS XI-2.6 Summary

Clear

SIS XI Section 3 SPT Impact on Estimating Methodology

SPS XI-3.1 Introduction

Clear

SPS XI-3.2 SP Impact on Similar Experience Technique

SPS XI-3.2.1 Model Description

Clear

SPS XI-3.2.2 Feasibility Type Estimates

Clear

SPS XI-3.2.3 Commitment Type Estimate

Clear

SPS XI-3.2.4 Operational Type Estimate

Clear

SPS XI-3.3 SP Impact on Quantitative Technique

Clear

SPS XI-3.4 SP Impact on Computer Resource Estimating Guidelines

Clear

SPS XI-3.5 Statistical Validation Needs

Clear

SPS XI Appendix A Software Development Cycle

SPS XI-A.1 Introduction

Clear

SPS XI-A.2 Development Phases

SPS XI-A.2.1 System Definition

Clear

SPS XI-A.2.2 System Design

Clear

SPS XI-A.2.3 System Implementation

Clear

SPS XI-A.2.4 System Evaluation

Clear

SPS XI-A.2.5 Relationship of Phases

Clear

SPS XI Appendix B Quantitative Estimating Technique

SPS XI-B.1 Introduction

Clear

SPS XI-B.2 Quantitative Technique

SPS XI-B.2.1 Definitions and Assumptions

The small contractor is not likely to be a prime contractor on a programming project that requires more than 25 programmers. The Quantitative Technique would not be applicable to his environment.

SPS XI-B.2.2 Factors Affecting the Estimating Technique

Clear

SPS XI-B.2.3 Quantitative Estimating Procedure

Clear

SPS XI Appendix C SPT Fundamentals

Clear

SPS XI Appendix D Definitions of Factors Influencing Estimates

SPS XI-D.1 Introduction

Clear

SPS XI-D.2 Definitions

SPS XI-D.2.1 Uniqueness Factors

Clear

SPS XI-D.2.2 Development Environment Factors

Clear

II.5 Program Documentation Standards

The subject of Program Documentation Standards is addressed in Volume VII, Documentation Standards, of the Structured Programming Series. This Chapter II.5 provides comments and recommendations concerning Program Documentation Standards.

SPS VII Documentation Standards

SPS VII Section 1 Introduction

SPS VII-1.1 Background

Clear

SPS VII-1.2 Report Organization

Clear

SPS VII-1.3 Conclusions

Clear

SPS VII-1.4 Recommendations

Clear

SPS VII Section 2 Advances in Programming Technology

SPS VII-2.1 Introduction

Clear

SPS VII-2.2 TDSP

Clear

SPS VII-2.2.1 SP

Clear

SPS VII-2.2.2 Top Down Programming

Clear

SPS VII-2.3 PSL

Clear

SPS VII-2.4 PDL

Clear

SPS VII-2.5 Definitions

Clear

SPS VII Section 3 Current Documentation Standards

SPS VII-3.1 Summary of Current Standards

SPS VII-3.1.1 MIL-STD-490

Clear

SPS VII-3.1.2 MIL-STD-483

Clear

SPS VII-3.1.3 DOD 4120.17M

Clear

SPS VII-3.2 Documents Produced

Clear

SPS VII-3.2.1 Document Types

Clear

SPS VII-3.2.2 Intended Audience

Clear

SPS VII-3.3 Document Content

Clear

SPS VII Section 4 Impact of SP Technology

SPS VII-4.1 Traditional Program Documentation

Clear

SPS VII-4.2 SP Technology Improvements

SPS VII-4.2.1 TDSP

Clear

It is recommended that printer forms be developed that have vertical lines so that the indentation of source code will be more useful. The beginning and end of a segment would be clearly visible especially when nesting of control structures occur.

SPS VII-4.2.2 PSL

Clear

This subsection is the first to mention that a one segment per page format should be adhered to. Assuming the programmer will be working from listings created by a precompiler, then the precompiler must be able to page the output according to a programmer-inserted directive. An additional feature of the precompiler is that it be able to distinguish when a segment has changed and only print those segments. No mention was made of either of these features when the precompiler was discussed in Volumes II and III.

SPS VII-4.2.3 PDL

Clear

SPS VII Section 5 Revised Documentation Systems

SPS VII-5.1 Specific Impact

Clear

SPS VII-5.2 Explanation of Appendices A and B

Clear

SPS VII-5.3 Explanation of Appendix C

Clear

SPS VII Appendix A Revisions to USAF MIL-STD-483

SPS VII-A.I Introduction

Clear

SPS VII-A.II Summary

Clear

SPS VII-A.III Proposed Revisions

Revision A1

Clear

SPS VII-Revision A2

Clear

SPS VII-Revision A3

Clear

SPS VII-Revision A4

Clear

SPS VII-Revision A5

Clear

SPS VII Appendix B Revisions to DoD 4120.17M

SPS VII-B.I Introduction

Clear

SPS VII-B.II Summary

Clear

SPS VII-B.III Proposed Revisions

SPS VII-Revision B1

Clear

SPS VII-Revision B2

Clear

SPS VII-Revision B3

Clear

SPS VII-Revision B4

Clear

SPS VII-Revision B5

Clear

SPS VII-Revision B6

Clear

SPS VII-Revision B7

Clear

SPS VII-Revision B8

Clear

SPS VII-Revision B9

Clear

SPS VII-Revision B10

Clear

SPS VII Appendix C Revisions to DoD 4120.17M

SPS VII-C.I Introduction

Clear

SPS VII-C.II Summary

Clear

SPS VII-C.III Proposed Revisions

SPS VII-Revision C1

Clear

SPS VII-Revision C2

Clear

SPS VII-Revision C3

Clear

SPS VII-Revision C4

Clear

SPS VII-Revision C5

Clear

SPS VII-Revision C6

Clear

SPS VII-Revision C7

Clear

SPS VII-Revision C8

Clear

SPS VII-Revision C9

Clear

SPS VII-Revision C10

Clear

SPS VII-Revision C11

Clear

SPS VII-Revision C12

Clear

SPS VII-Revision C13

Clear

SPS VII-Revision C14

Clear

SPS VII-Revision C15

Clear

SPS VII Section 6 Introduction

SPS VII-6.1 Background

Clear

SPS VII-6.2 Report Organization

Clear

SPS VII-6.3 Conclusions

Clear

SPS VII-6.4 Recommendations

Clear

SPS VII Section 7 Impact of SP Technology

SPS VII-7.1 Introduction

Clear

SPS VII-7.2 Recommended Revisions

SPS VII-7.2.1 AR 18-1

Clear

SPS VII-7.2.2 USACSC Manual 18-1 Volume 1

Clear

SPS VII-7.2.3 USACSC Manual 18-100

Clear

SPS VII Appendix D Revisions to AR 18-1

SPS VII-D.I Introduction

Clear

SPS VII-D.II Summary

Clear

SPS VII-D.III Proposed Revisions

SPS VII-Revision D1

Clear

SPS VII-Revision D2

Clear

SPS VII-Revision D3

Clear

SPS VII-Revision D4

Clear

SPS VII-Revision D5

Clear

SPS VII Appendix E Revisions to USACSC Manual 18-1

SPS VII-E.I Introduction

Clear

SPS VII-E.II Summary

Clear

SPS VII-E.III Proposed Revisions

SPS VII-Revision E1

Clear

SPS VII-Revision E2

Clear

SPS VII-Revision E3

Clear

SPS VII-Revision E4

Clear

SPS VII-Revision E5

Clear

SPS VII-Revision E6

Clear

SPS VII-Revision E7

Clear

SPS VII-Revision E8

Clear

SPS VII-Revision E9

Clear

SPS VII-Revision E10

Clear

SPS VII-Revision E11

Clear

SPS VII-Revision E12

Clear

SPS VII-Revision E13

Clear

SPS VII-Revision E14

Clear

SPS VII-Revision E15

Clear

SPS VII-Revision E16

Clear

Figure E-2 of Volume VII, Addendum is a HIPO diagram, the term should be introduced and used.

SPS VII-Revision E17

Clear

SPS VII-Revision E18

Clear

SPS VII-Revision E19

Clear

SPS VII-Revision E20

Clear

SPS VII-Revision E21

Clear

SPS VII-Revision E22

Clear

SPS VII Appendix F Revisions to USACSC Manual 18-100

SPS VII-F.I Introduction

Clear

SPS VII-F.II Summary

Clear

SPS VII-F.III Proposed Revisions

SPS VII-Revision F1

Clear

SPS VII-Revision F2

Clear

SPS VII-Revision F3

Clear

SPS VII-Revision F4

Clear

SPS VII-Revision F5

Clear

SPS VII-Revision F6

Clear

SPS VII-Revision F7

Clear

SPS VII-Revision F8

Clear

SPS VII-Revision F9

Clear

SPS VII-Revision F10

Clear

SPS VII-Revision F11

Clear

SPS VII-Revision F12

Clear

SPS VII-Revision F13

Clear

SPS VII-Revision F14

Clear

SPS VII-Revision F15

Clear

SPS VII-Revision F16

Clear

SPS VII-Revision F17

Clear

SPS VII-Revision F18

Clear

SPS VII-Revision F19

Clear

SPS VII-Revision F20

Clear

SPS VII-Revision F21

Clear

SPS VII-Revision F22

Clear

SPS VII-Revision F23

Clear

SPS VII-Revision F24

Clear

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.